

AMBROS GLEIXNER
STEPHEN J. MAHER
BENJAMIN MÜLLER
JOÃO PEDRO PEDROSO

Exact Methods for Recursive Circle Packing

Zuse Institute Berlin
Takustrasse 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Exact Methods for Recursive Circle Packing

Ambros Gleixner*

Stephen J. Maher*

Benjamin Müller*

João Pedro Pedroso†

February 28, 2017

Abstract

Packing rings into a minimum number of rectangles is an optimization problem which appears naturally in the logistics operations of the tube industry. It encompasses two major difficulties, namely the positioning of rings in rectangles and the recursive packing of rings into other rings. This problem is known as the Recursive Circle Packing Problem (RCPP). We present the first exact method for solving RCPP, based on a Dantzig-Wolfe decomposition of a nonconvex mixed-integer nonlinear programming formulation. The key idea of this reformulation is to break symmetry on each recursion level by enumerating all so-called one-level packings, i.e., packings of circles into other circles, and by dynamically generating packings of circles into rectangles. We propose a branch-and-price algorithm to solve the reformulation to global optimality. Extensive computational experiments on a large test set show that our method not only computes exact dual bounds, but often produces primal solutions better than computed by heuristics from the literature.

1 Introduction

Packing problems appear naturally in a wide range of real-world applications. They contain two major difficulties. First, complex geometric objects need to be selected, grouped, and packed into other objects, e.g., warehouses, containers, or parcels. Second, these objects need to be packed in a dense, non-overlapping way and, depending on the application, respect some physical constraints. Modeling packing problems mathematically often leads to nonconvex mixed-integer nonlinear programs (MINLPs). Solving them to global optimality is a challenging task for state-of-the-art MINLP solvers. In this paper, we will study exact algorithms for solving a particularly complex version involving the *recursive* packing of 2-dimensional rings. This variant has real-world applications in the tube industry.

The Recursive Circle Packing Problem (RCPP) has been introduced recently by Pedroso et al. [25]. The objective of RCPP is to select a minimum number of rectangles of the same size such that a given set of rings can be packed into these rectangles in a non-overlapping way. A ring is characterized by an internal and an external radius. Rings can be put recursively into larger ones or directly into a rectangle. A set of rings packed into a rectangle is called a *feasible packing* if and only if all rings lie within the boundary of the rectangle and do not intersect each other. Figure 1 gives two examples of a feasible packing.

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {benjamin.mueller,gleixner,maher}@zib.de

†Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal, jpp@ncc.up.pt

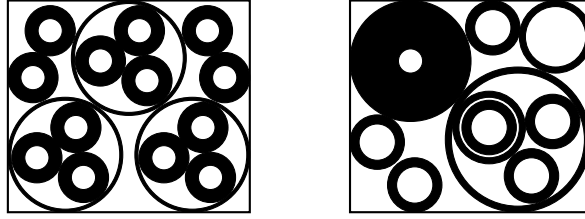


Figure 1: Two feasible packings of rings into rectangles.

Pedroso et al. [25] present a nonconvex MINLP formulation for RCPP. The key idea of the developed MINLP is to use binary variables in order to indicate whether a ring is packed inside another larger ring or directly into a rectangle. Due to a large number of binary variables and nonconvex quadratic constraints, the model is not of practical relevance and can only be used to solve very small instances. However, the authors present a well-performing local search heuristic to find good feasible solutions.

The purpose of this article is to present the first exact branch-and-price algorithm for RCPP based on a Dantzig-Wolfe decomposition, for which, so far, only heuristics exist. The new method is able to solve small- and medium-size instances to global optimality and prove strong primal and dual bounds for larger ones. We first develop a reformulation, which is similar to the classical reformulation for the Cutting Stock Problem [14], however, featuring nonlinear and nonconvex sub-problems. This formulation breaks the symmetry between equivalent rectangles. As a second step, we combine this reformulation with an enumeration scheme for patterns that are characterized by rings packed inside other rings. Such patterns only allow for a one-level recursion and break the symmetry between rings with the same internal and external radius in each rectangle. Finally, we present how to solve the resulting reformulation with a branch-and-price algorithm.

2 Background

The problem of finding dense packings of geometric objects has a rich history that goes back to Kepler’s Conjecture in 1611, which has been proven recently by Hales et al. [16]. Packing identical or diverse objects into different geometries like circles, rectangles, and polygons remains a relevant topic and has been the focus of much research during the last decades. The survey by Hifi and M’Hallah [17] reviews the most relevant results for packing 2- and 3-dimensional spheres into regions in the Euclidean space. Applications, heuristics, and exact strategies of packing arbitrarily sized circles into a container are presented by Castillo et al. [4]. Costa et al. [7] use a spatial branch-and-bound algorithm to find good feasible solutions for the case of packing identical circles. They propose different symmetry-breaking constraints to tighten the convex relaxation and improve the success rate of local nonlinear programming algorithms.

Closely related to packing problems are plate-cutting problems, in which convex objects, e.g., circles, rectangles, or polygons, are cut from different convex geometries [9]. Modeling these problems leads typically to nonlinear programs, for which exact mathematical programming solutions are described by Kallrath [20]. Minimizing the volume of a box for an overlap-free placement of ellipsoids was studied by Kallrath [21]. His closed, non-convex nonlinear programming formulation uses the entries of the rotation matrix as variables and can be used to get feasible solutions for instances with up to 100 ellipsoids.

The general type of the problem finds application in the tube industry, where shipping costs

represent a large fraction of the total costs of product delivery. Tubes will be cut to the same length of the container in which they may be shipped. In order to reduce idle space inside containers, smaller tubes might be placed inside larger ones. Packing tubes as densely as possible reduces the total number of containers needed to deliver all tubes and thus has a large impact on the total cost.

RCPP is a generalization of the well-known, \mathcal{NP} -hard [23] Circle Packing Problem (CPP) and therefore is also \mathcal{NP} -hard. Reducing an instance of CPP to RCPP can be done by setting all internal radii to zero, i.e., by forbidding to pack rings into larger rings. Typically, problems like RCPP contain multiple sources of symmetry. Any permutation of rectangles, i.e., relabeling rectangles, constitutes an equivalent solution to RCPP. Even worse, there is also considerable symmetry inside a rectangle. First, rotating or reflecting a rectangle gives an equivalent rectangle since both contain the same set of rings. Second, two rings with same internal and external radius can be exchanged arbitrarily inside a rectangle, again resulting in an equivalent rectangle packing.

One possible way to break the symmetry of RCPP is to add symmetry-breaking constraints, which have been frequently used for scheduling problems, e.g., lot-sizing problems [19]. An alternative approach is the use of decomposition techniques. These techniques aggregate identical sub-problems in order to reduce symmetry and typically strengthen the relaxation of the initial problem. A well-known decomposition technique is the Dantzig-Wolfe decomposition by Dantzig and Wolfe [13]. It is an algorithm to decompose Linear Programs (LPs) into a master and, in general, several sub-problems. Column generation is used with this decomposition to improve the solvability of large-scale linear programs. Embedded in a branch-and-bound algorithm, it can be used to solve mixed-integer linear programs (MILPs), e.g., bin packing [33], two dimensional bin packing [26], cutting stock [14, 15], and many other problems. Fortz et al. [11] applied a Dantzig-Wolfe decomposition to a stochastic network design problem with a convex nonlinear objective function. While most implementations of Dantzig-Wolfe are problem-specific, a number of frameworks have been implemented for automatically applying a Dantzig-Wolfe decomposition to a compact MILP formulation, see [3, 12, 27].

In this paper we apply a Dantzig-Wolfe decomposition to an MINLP formulation of RCPP and present the first exact solution method that is able to solve practically relevant instances. The rest of the paper is organized as follows. In Section 3, we introduce basic notation and discuss the limitations of a compact MINLP formulation for RCPP. Section 4 presents a first Dantzig-Wolfe decomposition of the MINLP formulation. After introducing the concept of circular and rectangular patterns, we extend the formulation from Section 4 in Section 5 to our final formulation for RCPP. Afterwards, we present a branch-and-price algorithm to solve this formulation, which uses an enumeration scheme introduced in Section 6. Section 7 shows how to prove valid dual and primal bounds, even when difficult sub-problems cannot be solved to optimality. Finally, in Section 8, we analyze the performance of our method on a large test set containing 800 synthetic instances and nine real-world instances from the tube industry. Section 9 gives concluding remarks.

3 Problem Statement

Consider a set $\mathcal{T} := \{1, \dots, T\}$ for T different types of rings and an infinite number of rectangles. In what follows, we consider each rectangle to be of size $W \in \mathbb{R}_+$ times $H \in \mathbb{R}_+$ and to be placed in the Euclidean plane such that the corners are $(0, 0)$, $(0, W)$, $(H, 0)$, and (W, H) .

For each ring type $t \in \mathcal{T}$ we are given an internal radius $r_t \in \mathbb{R}_+$ and an external radius $R_t \in \mathbb{R}_+$ such that

$$r_t \leq R_t \leq \min\{W, H\}.$$

Also, each ring type $t \in \mathcal{T}$ has a demand $D_t \in \mathbb{Z}_+$. We assume without loss of generality that \mathcal{T} is sorted such that $R_1 \leq \dots \leq R_T$. To simplify the notation, we denote by $n := \sum_{t \in \mathcal{T}} D_t$ the total number of individual rings and denote by $\mathcal{R} := \{1, \dots, n\}$ the corresponding index set. The function $\tau : \mathcal{R} \rightarrow \mathcal{T}$ maps each individual ring to its corresponding type. By a slight abuse of notation, we identify with r_i and R_i the internal and external radius of ring $i \in \mathcal{R}$, i.e., $R_i = R_{\tau(i)}$ and $r_i = r_{\tau(i)}$.

The task in RCPP is to pack all rings in \mathcal{R} into the smallest number of rectangles. Rings must lie within the boundary of a rectangle and must not intersect each other. More precisely, a feasible solution to RCPP can be encoded as a 3-tuple $(c, x, y) \in \{1, \dots, k\}^n \times \mathbb{R}^n \times \mathbb{R}^n$ where (x_i, y_i) denotes the center of ring $i \in \mathcal{R}$ inside rectangle $c_i \in \{1, \dots, k\}$, and an upper bound on the number of rectangles needed is given by $k \leq n$. The number of used rectangles is equal to the cardinality of $\{c_1, \dots, c_n\}$. Rings must not intersect the boundary of the rectangle, i.e.,

$$R_i \leq x_i \leq W - R_i \quad \text{for all } i \in \mathcal{R}, \quad (1)$$

$$R_i \leq y_i \leq H - R_i \quad \text{for all } i \in \mathcal{R}. \quad (2)$$

For a given 3-tuple (c, x, y) we denote by

$$A(i) := \left\{ (\tilde{x}, \tilde{y}) \in \mathbb{R}^2 \mid r_i < \left\| \begin{pmatrix} \tilde{x} - x_i \\ \tilde{y} - y_i \end{pmatrix} \right\|_2 < R_i \right\}$$

the area occupied by ring i in rectangle c_i . The non-overlapping condition between different rings is equivalent to

$$A(i) \cap A(j) = \emptyset \quad (3)$$

for all $i \neq j$ with $c_i = c_j$.

RCPP can be equivalently formulated as an MINLP, see Pedroso et al. [25]. The formulation consists of four different types of variables:

- $(x_i, y_i) \in \mathbb{R}^2$, the center of ring $i \in \mathcal{R}$,
- $z_c \in \{0, 1\}$, a decision variable whether rectangle $c \in \{1, \dots, k\}$ is used,
- $w_{i,c} \in \{0, 1\}$, a decision variable whether ring i is directly placed in rectangle c , and
- $u_{i,j} \in \{0, 1\}$, a decision variable whether ring i is directly placed in ring j .

We say that a ring $i \in \mathcal{R}$ is *directly placed* in another ring j or rectangle c if it is not contained in another, larger ring inside j or c , respectively. Condition (3) can be modeled by the constraints

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \geq (R_i + R_j)^2 (w_{i,c} + w_{j,c} - 1) \quad \forall i, j \in \mathcal{R} : i \neq j \quad \forall c \in \{1, \dots, k\}, \quad (4)$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \geq (R_i + R_j)^2 (u_{i,h} + u_{j,h} - 1) \quad \forall i, j, h \in \mathcal{R} : i \neq j \wedge i \neq h \wedge j \neq h, \quad (5)$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \leq r_j - R_i + M_{i,j}(1 - u_{i,j}) \quad \forall i, j \in \mathcal{R} : R_i \leq r_j, \quad (6)$$

which are nonconvex nonlinear inequality constraints. Inequality (4) guarantees that no two rings $i, j \in \mathcal{R}$ overlap when they are directly placed inside the same rectangle c , i.e., if $w_{i,c} = w_{j,c} = 1$. Similarly, Inequality (5) ensures that no two rings intersect inside another ring. Constraint (6) ensures that if $u_{i,j} = 1$, then ring i is directly placed inside j and does not

intersect its boundary. Otherwise, the constraint is disabled. An appropriate value for $M_{i,j}$ to guarantee that the conditions of (6) are satisfied is

$$M_{i,j} := \sqrt{(W - R_i - R_j)^2 + (H - R_i - R_j)^2},$$

which is the maximum distance between two rings $i, j \in \mathcal{R}$ in a W times H rectangle that do not overlap.

The full MINLP model for RCPP reads as follows:

$$\min \sum_{c=1}^k z_c \tag{7a}$$

$$\text{s.t. } w_{i,c} \leq z_c \quad \forall i \in \mathcal{R} \ \forall c \in \{1, \dots, k\} \tag{7b}$$

$$\sum_{c=1}^k w_{i,c} + \sum_{j \in \mathcal{R}} u_{i,j} = 1 \quad \forall i \in \mathcal{R} \tag{7c}$$

$$(1), (2), (4), (5), (6)$$

$$(x_i, y_i) \in \mathbb{R}^2 \quad \forall i \in \mathcal{R}$$

$$z_c \in \{0, 1\} \quad \forall c \in \{1, \dots, k\}$$

$$w_{i,c} \in \{0, 1\} \quad \forall i \in \mathcal{R} \ \forall c \in \{1, \dots, k\}$$

$$u_{i,j} \in \{0, 1\} \quad \forall i, j \in \mathcal{R} : i \neq j$$

Finally, the objective function (7a) minimizes the total number of rectangles used. Constraint (7b) guarantees that we can pack a ring inside a rectangle if and only if the rectangle is used. Each ring needs to be packed into another ring or a rectangle, which is ensured by (7c).

Remark 1 Formulation (7) can be adapted for maximizing the load of a single rectangle. Roughly speaking, we need to fix $z_c = 0$ for each $c > 1$ and replace each variable $w_{i,c}$ by w_i , which then turns into an indicator whether ring $i \in \mathcal{R}$ has been packed or not. The objective function (7a) changes to

$$\max \sum_{i \in \mathcal{R}} \alpha_i w_i,$$

where $\alpha_i \in \mathbb{R}_+$ is a non-negative weight for each ring $i \in \mathcal{R}$.

Unfortunately, general MINLP solvers require a considerable amount of time to solve Formulation (7) because it contains a large number of variables, many nonconvex constraints, and much symmetry. Even the smallest instance of our test set, containing 54 individual rings, could not be solved within several hours by SCIP 3.2.1 [29] nor by BARON 16.8.24 [22, 28]. A common method used to improve the solvability of geometric packing problems is to add symmetry breaking constraints. However, strengthening (7) by adding the constraints

$$z_c \geq z_{c+1} \quad \forall c \in \{1, \dots, k-1\}, \tag{8}$$

$$u_{i,j} \geq u_{h,j} \quad \forall j \in \mathcal{R} \ \forall i < h : \tau(i) = \tau(h), \text{ and} \tag{9}$$

$$w_{i,c} \geq w_{j,c} \quad \forall c \in \{1, \dots, k\} \ \forall i < j : \tau(i) = \tau(j), \tag{10}$$

which break symmetry by ordering rectangles and rings of the same type, had no impact on the solvability of RCPP.

The reason for the poor performance is that (7) contains $O(kn^2 + n^3)$ many difficult nonconvex constraints of the form (4) and (5). Even worse, the binary variables in those constraints appear with a big-M, which is known to result in weak linear programming relaxations. Another difficulty is the symmetry in the model that is not eliminated by (8), (9), or (10). Each reflection or rotation of a feasible packing of a rectangle and rings inside a ring yields another, equivalent, solution.

In the following, we present two formulations based on a Dantzig-Wolfe decomposition to break the remaining symmetry in (7). The first formulation breaks the symmetry between rectangles. The second is an extension of the first one and additionally breaks symmetry between rings of the same type inside the rectangles and other rings.

4 Cutting Stock Reformulation

Dantzig-Wolfe decomposition [13] is a classic solution approach for structured LPs. It can be used to decompose an MILP into a *master problem* and one or several *sub-problems*. Advantages of the decomposition are that it yields stronger relaxations if the sub-problems are nonconvex and can aggregate equivalent sub-problems to reduce symmetries [24, 8].

In this section, we apply a Dantzig-Wolfe decomposition to (7) and obtain a reformulation that is similar to the one of the one-dimensional Cutting Stock Problem [30]. The key idea is to reformulate RCPP in order to not assign rings explicitly to rectangles, but rather choose a set of feasible rectangle packings to cover the demand for each ring type. The resulting reformulation is a pure integer program (IP) containing exponentially many variables. We solve this reformulation via column generation.

We call a vector $F \in \mathbb{Z}_+^T$ *packable* if it is possible to pack F_t many rings for all types $t \in \mathcal{T}$ together (and thus a total number of $\sum_{t \in \mathcal{T}} F_t$ many rings) into a $W \times H$ rectangle. Denote by

$$\mathcal{F} := \{F \in \mathbb{Z}_+^T : F \text{ is packable}\}$$

the set of all feasible packings. Note that without loss of generality, we can bound F_t by the demand D_t for each $t \in \mathcal{T}$. After applying Dantzig-Wolfe decomposition to (7), we obtain the following formulation, $DW(\mathcal{F})$:

$$\min \sum_{F \in \mathcal{F}} z_F \tag{11a}$$

$$\text{s.t.} \quad \sum_{F \in \mathcal{F}} F_t \cdot z_F \geq D_t \quad \text{for all } t \in \{1, \dots, T\} \tag{11b}$$

$$z_F \in \mathbb{Z}_+ \quad \text{for all } F \in \mathcal{F} \tag{11c}$$

This formulation contains an exponential number of integer variables with respect to T . Each variable z_F counts how often F is chosen. Minimizing the total number of rectangles is equivalent to (11a). Inequality (11b) ensures that each ring of type t is packed at least D_t many times. In the following, we call the LP relaxation of $DW(\mathcal{F}')$ for $\mathcal{F}' \subseteq \mathcal{F}$ the *restricted master problem* of $DW(\mathcal{F})$.

An advantage of (11) is that we do not explicitly assign rings to positions inside some rectangles, like in (7), nor distinguish between rings of the same type. This breaks symmetry that arises from simply permuting rectangles. However, \mathcal{F} is a priori unknown and its size is exponential in the input size of an RCPP instance. One way to solve the LP relaxation of (11) is to use column generation, as outlined in Algorithm 1.

Algorithm 1: Column generation algorithm for solving the LP relaxation of $DW(\mathcal{F})$

in : initial set of feasible packings $\mathcal{F}' \subseteq \mathcal{F}$
out: optimal LP solution of $DW(\mathcal{F})$
1 (π^*, z^*) optimal primal-dual LP solution of $DW(\mathcal{F}')$
2 **while** $\exists F \in \mathcal{F} : \sum_{t \in \mathcal{T}} F_t \pi_t^* > 1$ **do**
3 choose F^1, \dots, F^ρ with $\sum_{t \in \mathcal{T}} F_t^i \pi_t^* > 1$ for $i = 1 \dots, \rho$
4 $\mathcal{F}' := \mathcal{F}' \cup \{F^1, \dots, F^\rho\}$
5 (π^*, z^*) optimal primal-dual LP solution of $DW(\mathcal{F}')$
6 **return** z^*

The pricing problem of Algorithm 1 is to find an $F \in \mathcal{F}$ such that the corresponding dual constraint

$$\sum_{t \in \mathcal{T}} F_t \pi_t^* \leq 1$$

is violated by the dual solution of the restricted master problem $DW(\mathcal{F}')$. The most violated dual constraint is found by solving

$$\min \left\{ 1 - \sum_{t \in \mathcal{T}} \pi_t^* F_t : F \in \mathbb{Z}_+^T, F \text{ is packable} \right\}. \quad (12)$$

This problem is a maximization variant of RCPP for which we need to find a subset of rings such that they can be packed into a single rectangle, see Remark 1. The objective gain of each ring of type t is exactly π_t^* . The LP relaxation of (11) is solved to optimality if the solution value of (12) is non-negative. Otherwise, we find an improving column that, after it has been added, might decrease the objective value of the restricted master problem $DW(\mathcal{F}')$.

As discussed in Remark 1, (12) can be modeled as a nonconvex MINLP that contains the selection and positioning of rings in a rectangle. Unfortunately, solving this problem is very difficult for a general MINLP solver. In our experiments none of the resulting pricing problems (12) could be solved to optimality.

5 Circular Pattern-based Dantzig-Wolfe Decomposition

The main drawback of (11) is that the resulting pricing problems (12) are intractable. This is due to two major difficulties, i) the positioning of rings inside a rectangle and ii) the combinatorial decisions of how to put rings into other rings. Together, they make the sub-problems much more difficult to solve than the IP master problem. The recursive decisions of packing rings into each other introduces much symmetry to (12). Rings of the same type, and all rings packed inside those, can be swapped inside a rectangle and yield an equivalent solution. This symmetry appears in each recursion level of the sub-problems. See Figure 2 for an example of this kind of symmetry.

The main idea of the following reformulation is to break this type of symmetry and shift the recursive decisions from the sub-problem to the master problem. This balances the complexity of both problems, which is crucial when using branch-and-price algorithms.

In the following, we introduce the concept of *circular* and *rectangular patterns*. These patterns describe possible packings of circles into rings or rectangles. The circles act like placeholders.

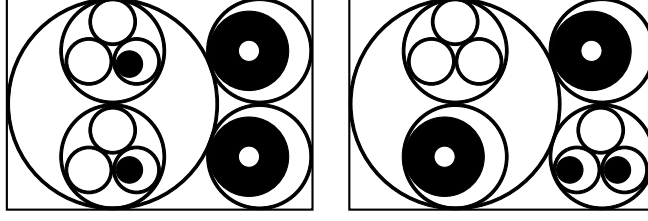


Figure 2: Two equivalent feasible packings obtained by swapping rings packed inside other rings.

Specifically, the circles just describe what type of rings might be placed in the circles, but not how these rings are filled with other rings. After choosing one pattern we are able to choose other patterns that fit into the circles of the selected one. The recursive part of RCPP boils down to a counting problem of patterns and minimizing the total number of rectangles.

5.1 Circular Patterns

A *circle* of type $t \in \mathcal{T}$ is a ring with external radius R_t and inner radius $r_t = 0$. This means that neither circles nor rings can be put into a circle. Similarly to the definition of elements in \mathcal{F} , we call a tuple $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ a *circular pattern* if it is possible to pack P_1 many circles of type 1, P_2 many circles of type 2, \dots , P_T many circles of type T into a larger ring of type t . Figure 3 shows all possible packings for three different ring types.

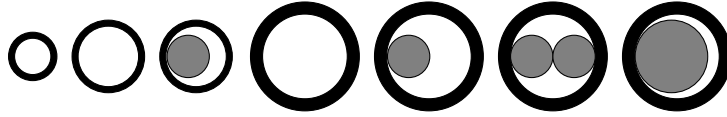


Figure 3: All possible circular patterns for three different ring types:
 $(1, (0, 0, 0)), (2, (0, 0, 0)), (2, (1, 0, 0)), (3, (0, 0, 0)), (3, (1, 0, 0)), (3, (2, 0, 0)), (3, (0, 1, 0))$

Using the definition of circular patterns we decompose a packing of rings into a ring R . Each ring that is directly placed into R , i.e., is not contained in another larger ring, is replaced by a circle with the same external radius. We apply the decomposition to each replaced ring recursively. As a result, ring R decomposes into a set of circular patterns. Starting from these circular patterns, we can reconstruct R by recursively replacing circles in a pattern with other circular patterns. We denote by

$$\mathcal{CP} := \{(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T \mid (t, P) \text{ is a circular pattern}\}$$

the set of all possible circular patterns. The previously described decomposition shows that any recursive packing of rings can be constructed by using circular patterns of \mathcal{CP} .

In general, the cardinality of \mathcal{CP} is exponential in T and depends on the internal and external radii of the rings. For example, increasing the inner radius of the largest ring will lead to many more possibilities to pack circles into this ring. In contrast, decreasing external radii results in fewer circular patterns.

Figure 3 shows that there are circular patterns that are dominated by others, e.g., the third pattern dominates second one. Using dominated circular patterns would leave some unnecessary free space in some rectangles, which can be avoided in an optimal solution. In Section 6 we

discuss this domination relation and present an algorithm to compute all non-dominated circular patterns.

5.2 Rectangular Patterns

In the following reformulation of RCPP, we use circular patterns to shift the decisions of how to put rings into other rings to the master problem. Similar to a circular pattern, we call $P \in \mathbb{Z}_+^T$ a *rectangular pattern* if and only if P_t many circles with radius R_t for each $t \in \mathcal{T}$ can be packed together into a rectangle of size W times H . Let

$$\mathcal{RP} := \{P \in \mathbb{Z}_+^T \mid P \text{ is a rectangular pattern}\}$$

be the set of all rectangular patterns. As in Formulation (11), only the number of packed circles matters, not their position in the rectangular pattern.

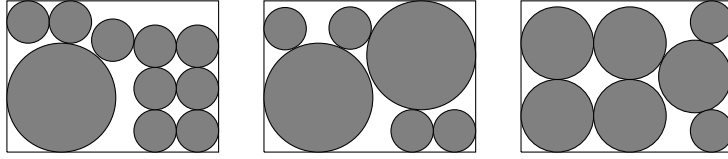


Figure 4: Three different rectangular patterns:
(9, 0, 1), (4, 0, 2), (2, 5, 0)

In contrast to verifying if $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ is a circular pattern, checking whether $P \in \mathbb{Z}_+^T$ is a rectangular pattern—a classical CPP—can be much more difficult. Typically, many more circles fit into a rectangle than into a ring. This results in a large number of circles that need to be considered in a verification problem, which is in practice difficult to solve.

5.3 Exploiting Recursion

Using the circle and rectangle patterns described above, we develop a pattern-based Dantzig-Wolfe decomposition for RCPP with nonlinear sub-problems—a major contribution of this paper. Instead of placing rings explicitly into each other, we use patterns to remodel the recursive part. The key idea is that circles, inside rectangular or circular patterns, are replaced by circular patterns with the same external radius. These circular patterns contain circles that can be replaced by other circular patterns. More precisely, after choosing a rectangular pattern $P \in \mathcal{RP}$, it is possible to choose P_t circular patterns of the form (t, P') in \mathcal{CP} , which can be placed into P . Again, for each P' we can choose P'_t many circular patterns of the form (t', P'') , which can be placed in P' . This process can continue until the smallest packable circle is considered. The recursive structure of the RCPP—the placement of rings into other rings—is modeled by counting the number of used rectangular and circular patterns.

Figure 5 illustrates this idea. A circular pattern replaces a circle if there is an arrow from the pattern to the circle. Each circle of a pattern can be used as often as the pattern is used. It follows that the number of outgoing edges of a circular pattern is equal to the number of uses of the pattern. The combinatorial part of RCPP reduces to adding edges from circular patterns to circles.

Following this idea, we introduce integer variables

- $z_C \in \mathbb{Z}_+$ for each circular pattern $C \in \mathcal{CP}$ and

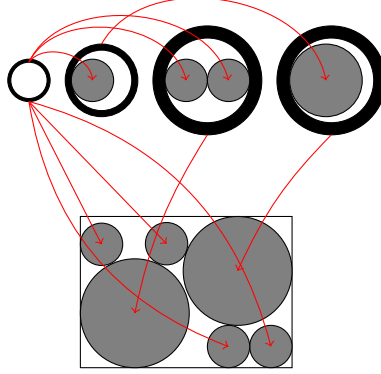


Figure 5: An example with four circular patterns and a rectangular pattern showing how patterns are used to model the combinatorial part of RCPP. Each line connects a circular pattern to a circle. The number of outgoing edges is equal to the number of rings that are used.

- $z_P \in \mathbb{Z}_+$ for each rectangular $P \in \mathcal{RP}$

in order to count the number of used circular and rectangular patterns (in the pattern-based formulation). We reformulate RCPP by the following IP formulation $PDW(\mathcal{RP})$:

$$\min \sum_{P \in \mathcal{RP}} z_P \quad (13a)$$

$$\text{s.t.} \quad \sum_{C=(t,P) \in \mathcal{CP}} z_C \geq D_t \quad \forall t \in \mathcal{T} \quad (13b)$$

$$\sum_{C=(t,P) \in \mathcal{CP}} z_C \leq \sum_{P \in \mathcal{RP}} P_t \cdot z_P + \sum_{C=(t',P) \in \mathcal{CP}} P_t \cdot z_C \quad \forall t \in \mathcal{T} \quad (13c)$$

$$z_C \in \mathbb{Z}_+ \quad \forall C \in \mathcal{CP} \quad (13d)$$

$$z_P \in \mathbb{Z}_+ \quad \forall P \in \mathcal{RP} \quad (13e)$$

Objective (13a) minimizes the total number of used rectangles. Constraint (13b) ensures that the demand for each ring type is satisfied. The recursive decisions how to place rings into each other are implicitly modeled by (13c). Each selection of a pattern allows us to choose P_t circular patterns of the type (t, P') . Note that at least one rectangular pattern needs to be selected before circular patterns can be packed. This is true because the largest ring only fits into a rectangular pattern.

The advantage of (13) is that it breaks the symmetry of the combinatorial part, i.e., packing rings into rings, on each recursion level. Deciding how to pack rings into each other is a counting problem in the master problem. Compared to (12), the resulting sub-problems do not contain the recursive structure of RCPP any more. This balances the complexity between the master problem and the sub-problems, which is crucial for the performance of a branch-and-price algorithm.

However, the drawback of (13) is the exponential number of rectangular and circular pattern variables. In Section 6 we present a column enumeration algorithm to compute all (relevant) circular patterns used in (13). We observed in our experiments that for many instances this algorithm could successfully enumerate all circular patterns in a reasonable amount of time.

In general, the size of the rectangles is be much larger than the external radii R , which makes an enumeration of all rectangular patterns intractable. To overcome this problem, we

use a column generation approach to solve the LP relaxation of (13) that generates rectangular patterns variables dynamically. As for Formulation (11), we call the LP relaxation of $PDW(\mathcal{RP}')$ the restricted master problem of (13) for a subset of rectangular patterns $\mathcal{RP}' \subseteq \mathcal{RP}$. In order to find an improving column for $PDW(\mathcal{RP}')$, we solve a weighted CPP for a single rectangle.

More precisely, let $\lambda \in \mathbb{R}_+^T$ be the non-negative vector of dual multipliers for Constraints (13c) after solving the LP relaxation of $PDW(\mathcal{RP}')$ for the current set of rectangular patterns $\mathcal{RP}' \subset \mathcal{RP}$. To compute a rectangular pattern with negative reduced costs we solve

$$\min_{P \in \mathcal{RP}} \left\{ 1 - \sum_{t \in \mathcal{T}} \lambda_t P_t \right\}, \quad (14)$$

which can be modeled as a weighted CPP for a single rectangle. This problem is \mathcal{NP} -hard [23] and difficult to solve in practice. The number of variables in this problem depends on the number of different ring types T and on the demand vector D . When solving (14) we need to consider the index set of individual circles

$$\{i_1^1, \dots, i_{D_1}^1, i_1^2, \dots, i_{D_2}^2, \dots, i_1^T, \dots, i_{D_T}^T\}$$

containing D_t indices that correspond to circles with radius R_t for each $t \in \mathcal{T}$. The number of copies for type t can be reduced to $\min\{D_t, \left\lfloor \frac{\pi(R_t)^2}{WH} \right\rfloor\}$, which is an upper bound on the number of rings of type t in a W times H rectangle.

Let P^* be an optimal solution to (14). If $1 - \sum_{t \in \mathcal{T}} \lambda_t P_t^*$ is negative, then P^* is an improving rectangular pattern, whose corresponding variable needs to be added to the restricted master problem of $PDW(\mathcal{RP}')$. Otherwise, we have solved the LP relaxation to optimality. In the latter case it might be necessary to branch on some variables with fractional solution values. We use the same branching strategy that has been introduced by [32] and that has furthermore been used in a branch-and-price algorithm for the One-dimensional Cutting Stock Problem [30]. The key idea is to search for a subset $\mathcal{P} \subseteq \mathcal{RP}$ such that $\sum_{P \in \mathcal{P}} z_P^*$ is fractional for the optimal LP solution z^* . Two new branching nodes are obtained by adding either

$$\sum_{P \in \mathcal{P}} z_P \leq \left\lfloor \sum_{P \in \mathcal{P}} z_P^* \right\rfloor \quad \text{or} \quad \sum_{P \in \mathcal{P}} z_P \geq \left\lceil \sum_{P \in \mathcal{P}} z_P^* \right\rceil.$$

Both sub-problems contain an additional constraint, whose dual multiplier needs to be considered in (14).

6 Enumeration of Circular Patterns

Formulation (13) contains one variable for each circular pattern in \mathcal{CP} . This set is, in general, of exponential size. We present a column enumeration algorithm to compute all relevant circular patterns for (13). The main step of the algorithm is to verify whether a given tuple $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ is in the set \mathcal{CP} or not. A tuple can be checked by solving the following nonlinear nonconvex verification problem:

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2 \geq R_i + R_j \quad \forall i, j \in C : i < j \quad (15a)$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right\|_2 \leq r_t - R_i \quad \forall i \in C \quad (15b)$$

$$x_i, y_i \in \mathbb{R} \quad \forall i \in C \quad (15c)$$

Here $C := \{1, \dots, \sum_i P_i\}$ is the index set of individual circles, and R_i the corresponding external radius of a circle $i \in C$. Model (15) checks whether all circles can be placed in a non-overlapping way into a ring of type $t \in \mathcal{T}$. Constraint (15a) ensures that no two circles overlap, and (15b) guarantees that all circles are placed inside a ring of type t .

The non-overlapping conditions (15a) are nonconvex and make (15) computationally demanding to solve. Due to the positioning of circles, (15) contains a lot of symmetry. Every rotation of a solution by 180° leads to another equivalent solution. Typically, this kind of symmetry is difficult to address within a global NLP solver. Branching on some continuous variables has most likely no impact on the dual bound. One way to overcome this problem is to break some symmetry of the problem by ordering circles of the same type in the x -coordinate non-decreasingly. We achieve this by adding

$$x_i \leq x_j \quad \forall i < j : R_i = R_j \quad (16)$$

to (15). Additionally, we add an auxiliary objective function $\min_{i \in C} x_i$ to (15). From our computational experiments we have seen that adding (16) to (15) makes it easier for the NLP solver to prove infeasibility.

As already seen in Section 5.1, there is a dominance relation between circular patterns, meaning that in any optimal solution of RCPP, dominated patterns can be replaced by non-dominated ones. Definition 1 formalizes this notion of a dominance relation between circular patterns.

Definition 1 A circular pattern $(t, P) \in \mathcal{CP}$ *dominates* $(t, P') \in \mathcal{CP}$ if and only if $P' <_{lex} P$, where $<_{lex}$ denotes the standard lexicographical order of vectors.

Let

$$\mathcal{CP}^* := \{(t, P) \in \mathcal{CP} \mid \nexists (t, P') \in \mathcal{CP} : (t, P') \text{ dominates } (t, P)\}$$

be the set of *non-dominated circular patterns*. The set \mathcal{CP}^* might be much smaller than \mathcal{CP} , but is in general of exponential size. Using \mathcal{CP}^* in (13), instead of the larger set \mathcal{CP} , results in fewer variables.

Finally, we present the procedure *EnumeratePatterns* in order to compute \mathcal{CP}^* . Algorithm 2 considers all possible $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ and checks whether (t, P) is a circular pattern by solving (15). The algorithm exploits the dominance relation between circular patterns to reduce the number of NLP solves and filter dominated patterns. In the following, we discuss the different steps of Algorithm 2 in more detail. For simplicity, we define the set $[x] := \{0, \dots, x\}$ for an integer number $x \in \mathbb{Z}_+$, and call a candidate pattern $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ *infeasible* if $(t, P) \notin \mathcal{CP}$ and feasible otherwise.

The algorithm maintains three sets \mathcal{CP}_{feas} , \mathcal{CP}_{infeas} , $\mathcal{CP}_{unknown}$, initialized to the empty set. In Line 3, we iterate through all possible pattern candidates (t, P) for a fixed $t \in \mathcal{T}$. In Line 5, we check whether P dominates a circular pattern already verified as infeasible and whether P is dominated by a circular pattern already verified as feasible. In both cases, P can be skipped. Otherwise, in Line 6, we solve a nonconvex verification NLP (15), which is the bottleneck of Algorithm 2. Roughly speaking, this NLP is easy to solve for the case where P , selected in Line 3, is component-wise too small or too large. In the first case, finding a feasible solution is easy due to a small number of circles. In the other case, we can conclude that the circles of the candidate P cannot be packed due to limited volume of the surrounding ring. The computationally expensive verification NLPs lie between these two extreme cases. Because of the two reversed dominance checks, it is in general unclear in which order the P in Line 3 should be enumerated. On the one hand, it can be beneficial to start with candidates that contain many circles. In case of a feasible candidate, many other candidates can be discarded because of the

Algorithm 2: EnumeratePatterns

```
in : internal and external radii  $r$  and  $R$ , demands  $D$ 
out:  $\mathcal{CP}_{feas} \subseteq \mathcal{CP}$ , unverified candidates  $\mathcal{CP}_{unknown}$ 
1  $\mathcal{CP}_{feas} := \emptyset, \mathcal{CP}_{infeas} := \emptyset, \mathcal{CP}_{unknown} := \emptyset$ 
2 for  $t \in \mathcal{T}$  do
3   for  $P \in [D_1] \times \dots \times [D_T]$  do
4     if  $\exists(P', t) \in \mathcal{CP}_{infeas} : P' <_{lex} P$  or  $\exists(P', t) \in \mathcal{CP}_{feas} : P' >_{lex} P$  then
5       continue
6      $status := \text{solve verification NLP (15)}$ 
7     if  $status = \text{"feasible"}$  then
8        $\mathcal{CP}_{feas} := \mathcal{CP}_{feas} \cup (P, t)$ 
9     if  $status = \text{"infeasible"}$  then
10       $\mathcal{CP}_{infeas} := \mathcal{CP}_{infeas} \cup (P, t)$ 
11     if  $status = \text{"time limit"}$  or  $\text{"memory limit"}$  then
12       $\mathcal{CP}_{unknown} := \mathcal{CP}_{unknown} \cup (P, t)$ 
13 filter all dominated patterns from  $\mathcal{CP}_{feas}$  and  $\mathcal{CP}_{unknown}$ 
14 return  $(\mathcal{CP}_{feas}, \mathcal{CP}_{unknown})$ 
```

dominance relation between circular patterns. On the other hand, an infeasible candidate that dominates another infeasible candidate can incur a redundant, difficult NLP (15) solve.

Algorithm 2 returns two sets of circular patterns. The first set contains all feasible circular patterns that could be successfully verified and is denoted by \mathcal{CP}_{feas} . The second set, $\mathcal{CP}_{unknown}$, contains all candidates that could not be verified because of working limits, e.g., a time or memory limit on the solve of (15). At the end, each non-dominated circular pattern is either in the set \mathcal{CP}_{feas} or $\mathcal{CP}_{unknown}$, which means that

$$\mathcal{CP}^* \subseteq \mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$$

holds.

Ideally, we have verified all candidates, i.e., $\mathcal{CP}_{unknown} = \emptyset$. However, since there are exponentially many candidates to check and each candidate requires to solve (15), it might not be possible to compute \mathcal{CP}^* in a reasonable amount of time.

Nevertheless, even if the set $\mathcal{CP}_{unknown}$ is non-empty, we are able to compute valid dual and primal bounds for RCPP. A valid primal bound is given when solving (13) after restricting the set of circular patterns to \mathcal{CP}_{feas} . This is because this restriction ensures that we only use packable patterns. Using $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ in (13) yields a valid relaxation of RCPP because we use at least all patterns in \mathcal{CP}^* and maybe some circular patterns that are not packable.

7 Computation of Valid Bounds

The Pricing Problem (14) is a classical CPP—an \mathcal{NP} -hard nonlinear optimization problem, which is in practice very hard to solve. State-of-the-art MINLP solvers might fail to prove global optimality for this type of problems. Nevertheless, the following theorem shows how to use a dual bound for (14) to compute a valid dual bound for RCPP.

Theorem 1 (Farley [10], Vance et al. [31]) Let ν_{RMP} be the optimum of the restricted master problem of $PDW(\mathcal{RP}')$, $\nu_{Pricing}$ be the optimal value for the Pricing Problem (14), and OPT be the optimal solution value of RCPP. Then the inequality

$$\left\lceil \frac{\nu_{RMP}}{1 - z_{Pricing}} \right\rceil \leq OPT$$

holds for all valid dual bounds $z_{Pricing} \leq \nu_{Pricing}$.

In our computational experiments we have seen that this bound can be indeed used to prove optimality in cases when the pricing problem could not be solved to optimality.

Another important part of our branch-and-price method is the initial step, when computing the set of non-dominated circular patterns \mathcal{CP}^* . Its computational cost depends on the number of different ring types T , the external radii R , and the internal radii r . Algorithm 2 returns two sets \mathcal{CP}_{feas} and $\mathcal{CP}_{unknown}$ with the properties

$$\begin{aligned} \mathcal{CP}_{feas} &\subseteq \mathcal{CP} \text{ and} \\ \mathcal{CP}^* &\subseteq \mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}. \end{aligned}$$

We use \mathcal{CP}_{feas} and $\mathcal{CP}_{unknown}$ to compute primal and dual bounds on RCPP. As discussed in Section 6, using \mathcal{CP}_{feas} leads to a valid primal bound because only verified circular patterns are used. In contrast, using $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ as input leads to a valid dual bound because $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ contains at least all circular patterns in \mathcal{CP}^* . Note that the latter also gives a valid primal bound if the solution includes no elements in $\mathcal{CP}_{unknown}$. The overall solution method is summarized in Algorithm 3.

Algorithm 3: RCPP Algorithm

```

in : internal and external radii  $r$  and  $R$ , demands  $D$ 
out: valid dual bound  $z_d$  and primal bound  $z_p$ 
1  $(\mathcal{CP}_{feas}, \mathcal{CP}_{unknown}) := EnumeratePatterns(r, R, D)$ 
2  $(z_d^1, z_p^1) := \text{branch-and-price with } \mathcal{CP}_{feas}$ 
3 if  $\mathcal{CP}_{unknown} = \emptyset$  then
4   | return  $(z_d^1, z_p^1)$ 
5 else
6   |  $(\mathcal{CP}_{feas}, \mathcal{CP}_{unknown}) := EnumeratePatterns(r, R, D)$     // different working limits
7   |  $(z_d^2, z_p^2) := \text{branch-and-price with } \mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ 
8   | return  $(z_d^2, z_p^1)$ 

```

In Line 2, Algorithm 3 solves the pattern-based Dantzig-Wolfe decomposition (13) with all circular patterns successfully verified during Algorithm 2. Due to working limits, we might fail to solve this RCPP instance to optimality and end up with a dual bound z_d^1 and primal bound z_p^1 . If $\mathcal{CP}_{unknown} = \emptyset$, then z_d^1 and z_p^1 are valid for the original problem. Otherwise, z_d^1 may not be valid and we call the branch-and-price algorithm again for $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$, which yields a valid dual bound z_d^2 . Instead of using \mathcal{CP}_{feas} and $\mathcal{CP}_{unknown}$ from Line 1 for the second call of the branch-and-price method, we call Algorithm 2 again in Line 6 with different working limits for solving each NLP (15).

8 Computational Experiments

In this section, we investigate the performance and the quality of the dual and primal bounds obtained by Algorithm 3 and analyze how they relate to specific properties of an instance. The branch-and-price algorithm presented in Section 5.3 is implemented in the MINLP solver SCIP [29]. We refer to [2, 34] for an overview of the general solving algorithm and MINLP features of SCIP.

8.1 Implementation

We extended SCIP by two plug-ins: one pricing plug-in for solving the LP relaxation of Formulation (13) and one plug-in for branching in the branch-and-price algorithm. Algorithm 2 is executed during the reading phase of an RCPP instance. To accelerate the verification of circular pattern candidates, we first apply a simple greedy heuristic before solving the expensive verification NLP (15). The heuristic places greedily each circle to the left-most position and proceeds until no circle fits anymore. Before solving the Pricing Problem (14), we first try to find improving rectangular patterns with the same greedy heuristic used for verifying circular patterns. We solve the pricing problem with a separate SCIP instance if the greedy heuristic fails to find an improving column.

8.2 Experimental Setup

We conducted three main experiments. In the first experiment we characterize instances for which Algorithm 2 finds all elements of the set of non-dominated circular patterns \mathcal{CP}^* . The second experiment answers the question whether our proposed method is able to solve instances to global optimality and characterizes these instances by their structural properties. Because our method can also be used as a primal heuristic, in the last experiment we compare it with the GRASP heuristic of Pedroso et al. [25].

In the enumeration experiment we apply Algorithm 2 on each instance and check whether \mathcal{CP}^* could be computed in two hours. To avoid expensive calls of the verification NLP (15), we use a simple greedy heuristic to check whether a given candidate (t, P) is a circular pattern, i.e., if $(t, P) \in \mathcal{CP}$. The heuristic iteratively packs circles to the left-most, and then lowest possible position in a ring of type $t \in \mathcal{T}$. If the heuristic fails to verify a candidate, we solve (15) until a feasible solution has been found, or it has been proven to be infeasible. For very difficult problems it might happen that we spend the whole time limit in solving a single NLP.

For our second experiment, we call Algorithm 3 to compute a valid primal bound in Line 1 and 2, and a valid dual bound in Line 6 and 7. We set a total time limit of two hours for the computation of each bound and use different settings for the enumeration of circular patterns in Line 1 and 6. In contrast to the first experiment, we enforce a time limit of $t_{NLP} \in \mathbb{R}_+$ for each NLP (15) until half of the total time limit has past. Afterwards, we stop solving (15) and only use a greedy algorithm to verify circular pattern candidates. A pattern is added to \mathcal{CP}_{feas} if it could be verified. Otherwise, we add a candidate to $\mathcal{CP}_{unknown}$ and process with the next candidate pattern. During branch-and-price, we use a simple greedy heuristic in order to find an improving rectangular pattern before solving the Pricing Problem (14) exactly. If the heuristic fails to compute a column with negative reduced cost, we use a separate SCIP instance to solve (14) with a time limit of $t_{CPP} \in \mathbb{R}_+$.

Depending on whether we compute a primal or a dual bound, we use different specifications for the branch-and-price method (in Line 2 and 7 of Algorithm 3) and Algorithm 2:

1. For computing a valid primal bound we use $t_{NLP} = 30s$ and $t_{CPP} = 600s$. If we fail to solve a pricing problem to optimality and no improving column could be found, then we stop solving any further pricing problems and continue to solve the restricted master problem for the current set of rectangular patterns. In our experiments, this only occurs directly at the root node. In this case our method can be considered as a price-and-branch algorithm.
2. When computing a valid dual bound, we use $t_{NLP} = 100s$ and $t_{CPP} = \infty$, which means that we spend more time to verify circular pattern candidates and use the remaining time to solve (14). We obtain a valid dual bound by applying Theorem 1 if the last pricing problem could not be solved. In this case we have reached the time limit and the solution process stops.

Finally, in our third experiment we compare the obtained primal bounds of our branch-and-price algorithm with the ones of the GRASP heuristic. Both algorithms run with a time limit of two hours on each instance. We skip the dual bound computation of Algorithm 3 in Line 7 and use the same specifications for the branch-and-price method and Algorithm 2 as in the primal bound computation of our second experiment. We use the Python implementation of GRASP from [25]. Note that SCIP is written in the programming language C in which an implementation of GRASP would be much faster. However, in our primal bound experiment we only compare the quality of obtained solutions and it can be observed that GRASP finds its best solutions in the first few minutes.

Test Sets. We consider two different test sets for our experiments. The first one contains 9 real-world instances from the tube industry, which were used in [25], and is in the following called REAL test set. Because this test set is too limited for a detailed computational study, we created a second test set containing 800 instances, the RAND test set. The purpose of this set is to show how the number of different ring types T , the maximum ratio of external radii $\max_t r_t / \min_t R_t$, and the ratio between rectangle size and the maximum volume of a ring $\max\{W, H\} / \max_t \pi(R_t)^2$ influence the performance of our branch-and-price method.

The name of each instance reads `i<T>_<α>_<β>_<γ>.rpa` where

- $T \in \{3, 4, 5, 10\}$ is the number of ring types,
- $\frac{\max_t r_t}{\min_t R_t} =: \alpha \in \{2.0, 2.3, \dots, 4.7\}$ is the maximum external radii ratio,
- $\frac{\max\{W, H\}}{\max_t R_t} =: \beta \in \{2.0, 2.3, \dots, 4.7\}$ is the rectangle size to external radius ratio, and
- $\frac{WH}{\max_t \pi(R_t)^2} [0.8\gamma, 1.2\gamma]$ for $\gamma \in \{5, 10\}$ is the demand interval for each type t .

The demand of a type $t \in \mathcal{T}$ is randomly chosen from the corresponding demand interval. The size of this interval is anti-proportional to the external radius, i.e., types with large external radius appear less often. For all instances in RAND we fixed the width W and height H of the rectangles to 10. We created one instance for each 4-tupel, giving 800 instances in total.

Hardware and Software. The experiments were performed on a cluster of 64bit Intel Xeon X5672 CPUs at 3.2GHz with 12MB cache and 48GB main memory. In order to safeguard against a potential mutual slowdown of parallel processes, we ran only one job per node at a time. We used SCIP developer version 3.2.1.1 with CPLEX 12.6.0.0 as LP solver [18], CppAD 20140000.1 [5], and Ipopt 3.12.4 with MUMPS 4.10.0 [1] as NLP solver [35, 6].

8.3 Computational Results

In the following, we discuss the results for the three described experiments in detail. Instance-wise results of all experiments can be found in Table 6 in the appendix.

Enumeration Experiments. Figure 6 shows the computing time required to compute all non-dominated circular patterns \mathcal{CP}^* for the instances of the RAND test set. Each line corresponds to the subset of instances with identical number of ring types T . Each point on a line is computed as the shifted geometric mean (with a shift of 1.0) over all instances that have the same value $\max_t r_t / \min_t R_t$. We expect that the number of circular patterns increases when increasing the ratio between the ring with largest inner and the ring with smallest outer radius.

The first observation is that the time to compute all circular patterns increases when T increases. For example, for $T = 10$ we need about 4–10 times longer to enumerate all patterns than for $T = 5$. Also, all lines in Figure 6 approximately increase exponentially in $\max_t r_t / \min_t R_t$. A larger ratio implies that each verification NLPs (15) has a larger number of circles that fit into a ring of inner radius $\max_t r_t$. These difficult NLPs and the larger number of possible circular patterns provide an explanation for the exponential increase of each line in Figure 6.

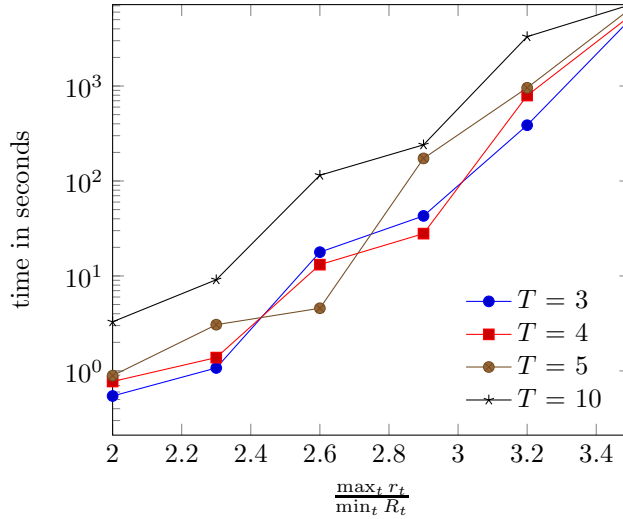


Figure 6: Plot showing the average time to enumerate all circular patterns for different number of ring types $T \in \{3, 4, 5, 10\}$ within a two hours time limit.

Table 1 shows in more detail for how many instances we could compute \mathcal{CP}^* successfully and how much time it took on average. The rows partition the instances according to the number of ring types and the columns group these instances by their $\max_t r_t / \min_t R_t$ values. First, we see that for 392 of the 800 instances of the RAND test set we could successfully enumerate all circular patterns, in 19.8 seconds on average. We see that for less and less instances the enumeration step is successful as $\max_t r_t / \min_t R_t$ increases. For none of the RAND instances with $\max_t r_t / \min_t R_t$ larger than 3.5 could \mathcal{CP}^* be successfully enumerated. Many verification NLPs for these instances are too difficult to solve and often consume the whole time limit of two hours, even for the case of three different ring types. However, for the 480 instances with $\max_t r_t / \min_t R_t \leq 3.5$ Algorithm 2 managed to compute on 81.6% (392) of these instances all non-dominated circular patterns.

Table 1: Aggregated results for enumerating all non-dominated circular patterns. Each of the first four rows contains 200 instances of the RAND test set.

n — number of instances for which \mathcal{CP}^* could be computed
time — time in seconds

| $\frac{\max_t r_t}{\min_t R_t}$ | ≥ 2.0 | | ≥ 2.3 | | ≥ 2.6 | | ≥ 2.9 | | ≥ 3.2 | | ≥ 3.5 | |
|---------------------------------|------------|------|------------|------|------------|-------|------------|-------|------------|--------|------------|--------|
| | n | time | n | time | n | time | n | time | n | time | n | time |
| $T = 3$ | 109 | 19.8 | 89 | 36.3 | 69 | 85.9 | 49 | 216.8 | 29 | 750.0 | 12 | 4050.7 |
| $T = 4$ | 104 | 17.0 | 84 | 30.3 | 64 | 69.7 | 44 | 177.8 | 24 | 912.7 | 9 | 4020.3 |
| $T = 5$ | 98 | 19.4 | 78 | 36.7 | 58 | 83.9 | 39 | 328.0 | 20 | 863.9 | 4 | 4507.7 |
| $T = 10$ | 81 | 24.7 | 61 | 46.1 | 41 | 104.4 | 25 | 235.0 | 8 | 1022.6 | 0 | 0.0 |
| all | 392 | 19.8 | 312 | 36.3 | 232 | 83.5 | 157 | 230.2 | 81 | 848.8 | 25 | 4109.4 |

Exact Branch-and-Price. In our second experiment, we use Algorithm 3 to obtain a primal bound by using \mathcal{CP}_{feas} , and a dual bound by using $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ for the set of circular patterns in Formulation (13). Figure 7 shows the achieved optimality gaps, i.e., $(\text{primal} - \text{dual})/\text{dual}$, for all instances of the RAND test set. We solve 37.2% of the instances to global optimality and get gaps between 0-25% for 5.8% of the instances. For about 53% of the instances we achieve gaps between 25-100%, and for only 4.0% gaps larger than 100%.

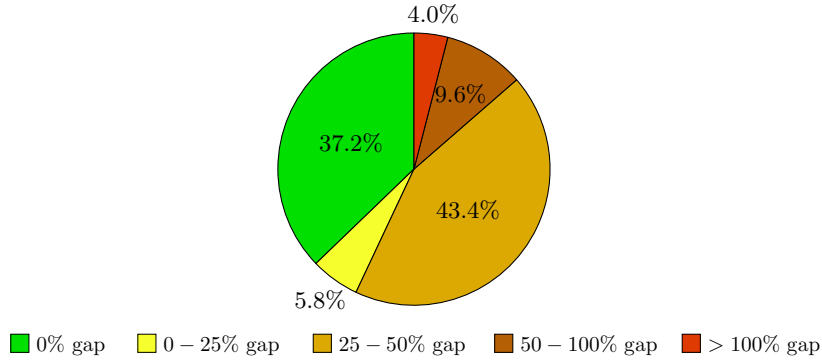


Figure 7: Optimality gaps for all RAND instances.

Table 2 and Table 3 contain aggregated results for the optimality gaps for the RAND test set. Table 2 shows that out of the 288 optimally solved instances, 86 instances have three, 68 have four, 74 have five, and 60 have ten different ring types. Most of the instances with a gap larger than 100% are from the subset of instances with ten ring types. Only three of the 31 instances with a gap larger than 100% have less than ten different ring types. Figure 8 visualizes the results of Table 2 in a bar diagram. As expected, for an increasing number of ring types worse optimality gaps are obtained.

Each row in Table 3 corresponds to the set of instances that have at least a certain value for $\max_t r_t / \min_t R_t$. For example, the bottom-left corner value signifies that 58 out of the 160 instances with $\max_t r_t / \min_t R_t \geq 4.4$ could be solved to optimality. As for Table 2, we see that the gaps increase with a larger $\max_t r_t / \min_t R_t$ ratio. This can be explained by our previous observations, namely, the difficulty of enumerating all circular patterns for these instances. On

Table 2: Number of instances of the RAND test set according to final gap and number of ring types.

| | total | | 0% | 0 – 25% | 25 – 50% | 50 – 100% | > 100% |
|----------|-------|--|-----|---------|----------|-----------|--------|
| $T = 3$ | 200 | | 86 | 25 | 79 | 10 | 0 |
| $T = 4$ | 200 | | 68 | 24 | 101 | 5 | 1 |
| $T = 5$ | 200 | | 74 | 21 | 92 | 11 | 2 |
| $T = 10$ | 200 | | 60 | 10 | 77 | 20 | 28 |
| all | 800 | | 288 | 80 | 349 | 46 | 31 |

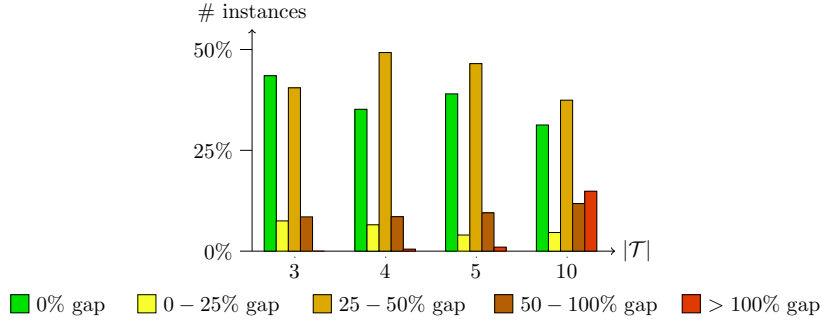


Figure 8: Optimality gaps for RAND instances split by number of different ring types.

the one hand, we might fail to find all circular patterns that are necessary for an optimal solution. On the other hand, we accept many circular patterns in $\mathcal{CP}_{unknown}$ that are not packable, which leads to poorer dual bounds. To summarize, our results show that the number of ring types T and the quotient $\max_t r_t / \min_t R_t$ are in many cases reliable indicators for both the quality of primal and dual bounds and the computational cost of our branch-and-price method. Instances that require branching are often too difficult to solve to global optimality.

Table 3: Number of instances of the RAND test set according to final gap and $\max_t r_t / \min_t R_t$.

| | total | | 0% | 0 – 25% | 25 – 50% | 50 – 100% | > 100% |
|------------------------------------|-------|--|-----|---------|----------|-----------|--------|
| $\max_t r_t / \min_t R_t \geq 2.0$ | 800 | | 288 | 80 | 349 | 46 | 31 |
| ≥ 2.6 | 640 | | 215 | 71 | 278 | 39 | 31 |
| ≥ 3.2 | 480 | | 165 | 50 | 194 | 35 | 30 |
| ≥ 3.8 | 320 | | 120 | 27 | 117 | 23 | 29 |
| ≥ 4.4 | 160 | | 58 | 12 | 59 | 11 | 17 |

Finally, we briefly report on the size of the branch-and-bound trees for the primal and dual bound experiments. For the dual bound runs the pricing problems (14) are computationally very demanding and consume for most of the instances the remaining time limit that is left after the initial enumeration and the call of the greedy heuristic. In this case the solution process stops at the root node. Only on 17 of the 800 RAND instances we process more than one node. The maximum number of nodes for those instances is 113. In the primal bound run this was the case

for seven instances with a maximum tree size of 201. Interestingly, only 12 instances could be solved to optimality that required branching either in the dual or primal bound run.

Primal Bound Experiments. In our final experiments, we consider our branch-and-price algorithm as a pure primal heuristic for RCPP and compare it to the GRASP heuristic by Pedroso et al. [25], which is specifically designed to find dense packings quickly. Table 4 contains detailed results for the REAL, and Table 5 aggregated results for the RAND test set. For analytical purposes, Table 4 also shows the results for the dual bounds.

Table 4: Detailed results for the comparison with GRASP on the REAL test set.

| Instance | | | branch+price | | GRASP |
|----------|----|---------------------------------|--------------|--------|--------|
| Name | T | $\frac{\max_t r_t}{\min_t R_t}$ | dual | primal | primal |
| s03i1 | 3 | 2.2 | 1 | 1 | 1 |
| s03i2 | 3 | 2.2 | 9 | 10 | 10 |
| s03i3 | 3 | 2.2 | 82 | 100 | 95 |
| s05i1 | 5 | 5.1 | 1 | 1 | 1 |
| s05i2 | 5 | 5.1 | 8 | 10 | 10 |
| s05i3 | 5 | 5.1 | 73 | 94 | 98 |
| s16i1 | 16 | 6.3 | 1 | 1 | 1 |
| s16i2 | 16 | 6.3 | 8 | 10 | 10 |
| s16i3 | 16 | 6.3 | – | 96 | 96 |

For the REAL test set, we are able to solve the three instances **s**i1**, that require only one rectangle in the optimal solution. On all but two instances branch-and-price achieves the same primal bound as GRASP. On **s03i3** we need five more rectangles and on **s05i3** four rectangles less. Due to the small value $\max_t r_t / \min_t R_t = 2.2$ of the instances with three different ring types, the enumeration step takes only a fraction of a second. The difficulty of these instances lies only in the placement of rings into rectangles and not in the recursive structure of RCPP. Hence, it is not surprising that our method computes a worse solution than GRASP on **s03i3** because we only use a simple left-most greedy heuristic before solving (14).

Regarding the proof of optimality, the instances of the REAL test set are difficult for our method. Only the three instances **s**i1** with one rectangle in the optimal packing, could be solved to gap zero. On the one hand, because of too large rectangles relative to the radii of the rings, the pricing problems contain many variables and constraints. The resulting problems are too difficult to be solved to optimality. On the other hand, as shown above, the ratio $\max_t r_t / \min_t R_t$ of the instances with $T > 3$ is too large to enumerate and verify all circular patterns in reasonable time. For **s16i3** no dual bound could be computed, because our method hit the memory limit during the enumeration of circular patterns.

However, even though our method is not particularly designed for the characteristics of the instances in the test set, it still performs well as a primal heuristic. On the instances with five ring types it could enumerate many non-trivial circular patterns and uses them to find a better primal solution than GRASP.

For the RAND test set, Table 5 shows the number of instances on which our method performed better or worse than GRASP with respect to the achieved primal bound. The second column reports the primal bound average relative to the value of GRASP on all instances of RAND. For this we compute the shifted geometric mean of all ratios between the obtained primal bounds of our method and GRASP. The third column shows the total number of instances on which our

method found a better primal solution. The fourth column shows the improvement relative to GRASP. The remaining three columns show statistics for the instances on which our method performed worse.

As a notable result, we observe that our method outperforms GRASP on many instances. On average, the solutions are 2.4% better than the ones from GRASP. On the 126 instances with ten types the relative improvement is around 1.3% to 1.9% better than for instances with fewer ring types. In total, we find a better primal bound on 374 of the instances, and a worse bound on only 44 instances. Only on a single instance no primal solution could be found, due to a too expensive enumeration step. For ten ring types the average deterioration is 18.4%, which is considerably worse than the relative improvement of 7.0%. However, on this subset of RAND, we obtain a better primal solution for 126 instances and a worse solution only for 12 instances.

There are two reasons why Algorithm 3 performs better than GRASP on many instances. The first is that GRASP considers and positions each ring individually. A large demand vector D results in a large number of rings, which makes it difficult for GRASP to find good primal solutions. In contrast, the number of rings and circles that need to be considered in our pricing problems and in the enumeration is typically bounded by a number derived from some volume arguments instead of the entries of the demand vector D . Thus, scaling up D typically does not have a large impact when applying our branch-and-price algorithm as a primal heuristic. The second reason for the better performance is that for a given set of circular and rectangular patterns, the master problem takes the best decisions of how to pack rings recursively into each other. For instances where this part of the problem is difficult, we expect that our method performs better than GRASP. Indeed, on instances with a large number of ring types branch-and-price frequently finds better solutions than GRASP.

Table 5: Aggregated results for primal bound comparisons against GRASP on the RAND test set.

| | all — all instances | | | | | |
|----------|---|--------|------|-------|-------|---------|
| | better — instances with a better primal bound than GRASP | | | | | |
| | worse — instances with a worse primal bound than GRASP | | | | | |
| | no sol. — number of instances for which no feasible solution could be found | | | | | |
| | # — number of instances | | | | | |
| | % — average primal bound relative to average primal bound of GRASP | | | | | |
| | all | better | | worse | | |
| | % | # | % | # | % | no sol. |
| $T = 3$ | 98.4 | 62 | 94.3 | 7 | 107.0 | 0 |
| $T = 4$ | 97.5 | 87 | 93.5 | 11 | 107.0 | 0 |
| $T = 5$ | 97.8 | 99 | 94.4 | 14 | 108.4 | 0 |
| $T = 10$ | 96.5 | 126 | 93.0 | 12 | 118.4 | 1 |
| all | 97.6 | 374 | 93.7 | 44 | 110.5 | 1 |

In summary, our experiments on synthetic and real-world instances, using an implementation of the MINLP solver SCIP, show that Algorithm 3 solves small and medium-sized instances to global optimality. This was the case on 37.2% of the randomly generated instances and for three of nine real-world instances. Due to a costly enumeration preprocessing step and difficult sub-problems, our method is not able to solve larger instances, but still achieves good primal and dual bounds. Compared to the state-of-the-art heuristic for RCPP, our method finds on 46.8% of the instances better, and only on 5.4% of the instances worse primal solutions.

9 Conclusion

In this article, we have presented the first exact algorithm for solving practically relevant instances for the extremely challenging recursive circle packing problem. Our method is based on a Dantzig-Wolfe decomposition of a nonconvex MINLP model. The key idea of solving this decomposition via branch-and-price is to break symmetry of the problem by using circular and rectangular patterns. These patterns are used to model all possible combinations of packing rings into other rings. As a result, we were able to reduce the complexity of the sub-problems significantly and shift the recursive part of RCPP to a linear master problem.

In some sense, this reformulation can be interpreted as a reduction technique from RCPP to CPP. All occurring sub-problems in the enumeration of circular patterns and the pricing problems are classical CPPs and they constitute the major computational bottleneck. Every primal or dual improvement for this problem class would directly translate to better primal and dual bounds for RCPP. Still, in order to prove optimality it is usually necessary to solve the \mathcal{NP} -hard CPP to optimality, which can fail for harder instances. However, even for this case, the application of Theorem 1 and the pessimistic and optimistic enumeration of circular patterns guarantees valid primal and dual bounds for RCPP. The combination of column generation with column enumeration could be of more general interest when using decomposition techniques for MINLPs that lead to hard nonconvex sub-problems.

Finally, our current proof-of-concept implementation could certainly be improved further. To mention only one point, we currently use a rather simple greedy heuristic in order to find quickly a feasible solution for the verification NLP (15) and the Pricing Problem (14). By using more sophisticated heuristics for the well-studied CPP, we might be able to compute better optimality gaps for instances where the positioning of circles into rectangles or rings is difficult. Surprisingly, even with a simple greedy heuristic our method works well as a primal heuristic and finds for many instances better solutions than GRASP [25].

Acknowledgments

This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibility for the content of this publication is assumed by the authors.

References

- [1] MUMPS, Multifrontal Massively Parallel sparse direct Solver. <http://mumps.enseeiht.fr>
- [2] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007). [URN:nbn:de:kobv:83-opus-16117](https://nbn-resolving.org/urn:nbn:de:kobv:83-opus-16117)
- [3] Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M.E., Malaguti, E., Traversi, E.: Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming* **149**(1-2), 391–424 (2015)
- [4] Castillo, I., Kampas, F.J., Pintér, J.D.: Solving circle packing problems by global optimization: Numerical results and industrial applications. *European Journal of Operational Research* **191**(3), 786–802 (2008). URL <http://EconPapers.repec.org/RePEc:eee:ejores:v:191:y:2008:i:3:p:786-802>

- [5] COIN-OR: CppAD, a package for differentiation of CPP algorithms. <http://www.coin-or.org/CppAD>
- [6] COIN-OR: Ipopt, Interior point optimizer. <http://www.coin-or.org/Ipopt>
- [7] Costa, A., Hansen, P., Liberti, L.: On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square. *Discrete Applied Mathematics* **161**(1–2), 96 – 106 (2013). DOI <http://dx.doi.org/10.1016/j.dam.2012.07.020>. URL <http://www.sciencedirect.com/science/article/pii/S0166218X12002855>
- [8] Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, vol. 5. Springer Science & Business Media (2006)
- [9] DROR, M.: Polygon plate-cutting with a given order. *IIE Transactions* **31**(3), 271–274 (1999). DOI 10.1023/A:1007606007369. URL <http://dx.doi.org/10.1023/A:1007606007369>
- [10] Farley, A.A.: A note on bounding a class of linear programming problems, including cutting stock problems. *Oper. Res.* **38**(5), 922–923 (1990). DOI 10.1287/opre.38.5.922. URL <http://dx.doi.org/10.1287/opre.38.5.922>
- [11] Fortz, B., Labbé, M., Poss, M.: A branch-and-cut-and-price framework for convex minlp applied to a stochastic network design problem. In: *Workshop on Mixed Integer Nonlinear Programming*, p. 131 (2010)
- [12] Gamrath, G.: Generic branch-cut-and-price. Master’s thesis (2010)
- [13] George B. Dantzig, P.W.: Decomposition principle for linear programs. *Operations Research* **8**(1), 101–111 (1960). URL <http://www.jstor.org/stable/167547>
- [14] Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem. *Operations Research* **9** (1961)
- [15] Gilmore, P., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. *Operations research* **13**(1), 94–120 (1965)
- [16] Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. *arXiv* **1501.02155** (2015)
- [17] Hifi, M., M’Hallah, R.: A literature review on circle and sphere packing problems: Models and methodologies. *Adv. Operations Research* **2009**, 150,624:1–150,624:22 (2009). URL <http://dblp.uni-trier.de/db/journals/advor/advor2009.html#HifiM09>
- [18] ILOG, Inc: ILOG CPLEX: High-performance software for mathematical programming and optimization. See <http://www.ilog.com/products/cplex/>
- [19] Jans, R.: Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. *INFORMS Journal on Computing* **21**(1), 123–136 (2009)
- [20] Kallrath, J.: Cutting circles and polygons from area-minimizing rectangles. *Journal of Global Optimization* **43**(2), 299–328 (2009). DOI 10.1007/s10898-007-9274-6. URL <http://dx.doi.org/10.1007/s10898-007-9274-6>

- [21] Kallrath, J.: Packing ellipsoids into volume-minimizing rectangular boxes. *Journal of Global Optimization* pp. 1–35 (2015). DOI 10.1007/s10898-015-0348-6. URL <http://dx.doi.org/10.1007/s10898-015-0348-6>
- [22] Kiliç, M., Sahinidis, N.V.: Solving MINLPs with BARON. Talk at MINLP 2014, Carnegie Mellon University, Pittsburgh, PA, USA (2014). <http://minlp.cheme.cmu.edu/2014/papers/kilinc.pdf>
- [23] Lenstra, J.K., Kan, A.H.G.R.: Complexity of packing, covering and partitioning problems. *Econometric Institute* (1979)
- [24] Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. *Operations Research* **53**(6), 1007–1023 (2005)
- [25] Pedroso, J.P., Cunha, S., Tavares, J.N.: Recursive circle packing problems. *International Transactions in Operational Research* **23**(1-2), 355–368 (2016). DOI 10.1111/itor.12107. URL <http://dx.doi.org/10.1111/itor.12107>
- [26] Pisinger, D., Sigurd, M.: The two-dimensional bin packing problem with variable bin sizes and costs. *Discret. Optim.* **2**(2), 154–167 (2005). DOI 10.1016/j.disopt.2005.01.002. URL <http://dx.doi.org/10.1016/j.disopt.2005.01.002>
- [27] Ralphs, T.K., Galati, M.V.: Decomposition in integer linear programming. *Integer Programming: Theory and Practice* **3** (2005)
- [28] Sahinidis, N.V.: Baron: A general purpose global optimization software package. *Journal of Global Optimization* **8**(2), 201–205 (1996). DOI 10.1007/BF00138693. URL <http://dx.doi.org/10.1007/BF00138693>
- [29] SCIP – Solving Constraint Integer Programs. <http://scip.zib.de>
- [30] Vance, P.H.: Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications* **9**(3), 211–228 (1998). DOI 10.1023/A:1018346107246. URL <http://dx.doi.org/10.1023/A:1018346107246>
- [31] Vance, P.H., Barnhart, C., Johnson, E.L., Nemhauser, G.L.: Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications* **3**(2), 111–130 (1994). DOI 10.1007/BF01300970. URL <http://dx.doi.org/10.1007/BF01300970>
- [32] Vanderbeck, F., Wolsey, L.A.: An exact algorithm for ip column generation. *Oper. Res. Lett.* **19**(4), 151–159 (1996). DOI 10.1016/0167-6377(96)00033-8. URL [http://dx.doi.org/10.1016/0167-6377\(96\)00033-8](http://dx.doi.org/10.1016/0167-6377(96)00033-8)
- [33] Vanderbeck, F., Wolsey, L.A.: Reformulation and Decomposition of Integer Programs, pp. 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). DOI 10.1007/978-3-540-68279-0_13. URL http://dx.doi.org/10.1007/978-3-540-68279-0_13
- [34] Vigerske, S.: Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming. Ph.D. thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II (2013). URN:nbn:de:kobv:11-100208240

- [35] Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* **106**(1), 25–57 (2006). [DOI:10.1007/s10107-004-0559-y](https://doi.org/10.1007/s10107-004-0559-y)

Appendix

Table 6: Detailed results for randomly generated instances.

primal — proven dual bound

dual — proven primal bound

GRASP — proven primal bound by GRASP heuristic

n_{cp} — total number of circular patterns used for primal bound run

n_{cd} — total number of circular patterns used for dual bound run

$tree_p$ — total number of branch-and-bound nodes for primal bound run

$tree_d$ — total number of branch-and-bound nodes for dual bound run

enum time — time to compute all circular patterns in enumeration experiment

| instance | primal | dual | GRASP | n_{cp} | n_{cd} | $tree_p$ | $tree_d$ | enum time |
|----------------|--------|------|-------|----------|----------|----------|----------|-----------|
| i03.2.0.2.0_05 | 17 | 17 | 17 | 4 | 4 | 1 | 1 | 0.36 |
| i03.2.0.2.0_10 | 38 | 38 | 38 | 5 | 5 | 1 | 1 | 1.35 |
| i03.2.0.2.3_05 | 16 | 16 | 16 | 4 | 4 | 1 | 1 | 0.49 |
| i03.2.0.2.3_10 | 45 | 49 | 49 | 4 | 4 | 1 | 1 | 0.87 |
| i03.2.0.2.6_05 | 23 | 23 | 23 | 3 | 3 | 1 | 1 | 0.62 |
| i03.2.0.2.6_10 | 41 | 41 | 41 | 4 | 4 | 1 | 1 | 0.34 |
| i03.2.0.2.9_05 | 16 | 16 | 16 | 4 | 4 | 1 | 1 | 0.38 |
| i03.2.0.2.9_10 | 18 | 27 | 27 | 4 | 4 | 1 | 1 | 0.71 |
| i03.2.0.3.2_05 | 23 | 23 | 23 | 4 | 4 | 1 | 1 | 0.56 |
| i03.2.0.3.2_10 | 23 | 38 | 38 | 4 | 4 | 1 | 1 | 0.97 |
| i03.2.0.3.5_05 | 21 | 21 | 21 | 3 | 3 | 1 | 1 | 0.72 |
| i03.2.0.3.5_10 | 22 | 32 | 33 | 4 | 4 | 1 | 1 | 0.45 |
| i03.2.0.3.8_05 | 11 | 14 | 15 | 4 | 4 | 1 | 1 | 0.37 |
| i03.2.0.3.8_10 | 23 | 31 | 33 | 4 | 4 | 1 | 1 | 0.35 |
| i03.2.0.4.1_05 | 13 | 17 | 18 | 4 | 4 | 1 | 1 | 0.34 |
| i03.2.0.4.1_10 | 18 | 24 | 24 | 4 | 4 | 1 | 1 | 0.36 |
| i03.2.0.4.4_05 | 10 | 14 | 15 | 4 | 4 | 1 | 1 | 0.35 |
| i03.2.0.4.4_10 | 20 | 25 | 27 | 4 | 4 | 1 | 1 | 0.34 |
| i03.2.0.4.7_05 | 18 | 18 | 19 | 3 | 3 | 1 | 1 | 0.58 |
| i03.2.0.4.7_10 | 21 | 28 | 30 | 4 | 4 | 1 | 1 | 0.44 |
| i03.2.3.2.0_05 | 15 | 15 | 15 | 3 | 3 | 1 | 1 | 2.18 |
| i03.2.3.2.0_10 | 36 | 36 | 36 | 4 | 4 | 1 | 1 | 0.51 |
| i03.2.3.2.3_05 | 13 | 18 | 18 | 4 | 4 | 1 | 1 | 0.88 |
| i03.2.3.2.3_10 | 32 | 32 | 32 | 4 | 4 | 1 | 1 | 1.24 |
| i03.2.3.2.6_05 | 12 | 17 | 18 | 4 | 4 | 1 | 1 | 0.77 |
| i03.2.3.2.6_10 | 35 | 35 | 38 | 4 | 4 | 1 | 1 | 1.03 |
| i03.2.3.2.9_05 | 17 | 17 | 17 | 4 | 4 | 1 | 1 | 0.57 |
| i03.2.3.2.9_10 | 66 | 66 | 66 | 3 | 3 | 1 | 1 | 2.15 |
| i03.2.3.3.2_05 | 11 | 16 | 16 | 4 | 4 | 1 | 1 | 0.67 |
| i03.2.3.3.2_10 | 38 | 38 | 38 | 4 | 4 | 1 | 1 | 0.86 |
| i03.2.3.3.5_05 | 23 | 23 | 23 | 3 | 3 | 1 | 1 | 0.9 |
| i03.2.3.3.5_10 | 21 | 28 | 29 | 4 | 4 | 1 | 1 | 0.88 |
| i03.2.3.3.8_05 | 11 | 14 | 15 | 4 | 4 | 1 | 1 | 0.69 |
| i03.2.3.3.8_10 | 25 | 35 | 35 | 5 | 5 | 1 | 1 | 2.65 |
| i03.2.3.4.1_05 | 10 | 12 | 13 | 4 | 4 | 1 | 1 | 1.07 |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------------------|------------------------------|-------------------|-------------------|-----------|
| i03.2.3.4.1_10 | 29 | 31 | 30 | 3 | 3 | 1 | 1 | 0.92 |
| i03.2.3.4.4_05 | 11 | 15 | 15 | 4 | 4 | 1 | 1 | 0.5 |
| i03.2.3.4.4_10 | 19 | 24 | 25 | 5 | 5 | 1 | 1 | 1.63 |
| i03.2.3.4.7_05 | 18 | 22 | 20 | 3 | 3 | 1 | 1 | 0.84 |
| i03.2.3.4.7_10 | 23 | 30 | 30 | 4 | 4 | 1 | 1 | 0.83 |
| i03.2.6.2.0_05 | 15 | 20 | 19 | 4 | 4 | 1 | 1 | 1.47 |
| i03.2.6.2.0_10 | 34 | 34 | 34 | 4 | 4 | 1 | 1 | 1.34 |
| i03.2.6.2.3_05 | 11 | 16 | 14 | 4 | 4 | 1 | 1 | 1.61 |
| i03.2.6.2.3_10 | 33 | 39 | 40 | 4 | 4 | 1 | 1 | 1.44 |
| i03.2.6.2.6_05 | 25 | 25 | 25 | 3 | 3 | 1 | 1 | 2.25 |
| i03.2.6.2.6_10 | 26 | 36 | 37 | 7 | 7 | 1 | 4 | 11.58 |
| i03.2.6.2.9_05 | 30 | 30 | 30 | 3 | 3 | 1 | 1 | 3.23 |
| i03.2.6.2.9_10 | 67 | 67 | 67 | 3 | 3 | 1 | 1 | 2.52 |
| i03.2.6.3.2_05 | 19 | 19 | 19 | 4 | 4 | 1 | 1 | 1.64 |
| i03.2.6.3.2_10 | 37 | 37 | 37 | 6 | 6 | 1 | 1 | 1599.43 |
| i03.2.6.3.5_05 | 12 | 16 | 17 | 4 | 4 | 1 | 1 | 2.03 |
| i03.2.6.3.5_10 | 24 | 34 | 35 | 6 | 7 | 1 | 1 | 5686.42 |
| i03.2.6.3.8_05 | 8 | 11 | 11 | 4 | 4 | 1 | 1 | 1.7 |
| i03.2.6.3.8_10 | 38 | 38 | 38 | 3 | 3 | 1 | 1 | 1.98 |
| i03.2.6.4.1_05 | 9 | 12 | 12 | 5 | 5 | 1 | 1 | 4.63 |
| i03.2.6.4.1_10 | 24 | 26 | 26 | 3 | 3 | 1 | 1 | 2.52 |
| i03.2.6.4.4_05 | 15 | 15 | 15 | 3 | 3 | 1 | 1 | 1.91 |
| i03.2.6.4.4_10 | 22 | 31 | 32 | 6 | 7 | 1 | 1 | 1498.95 |
| i03.2.6.4.7_05 | 10 | 14 | 14 | 4 | 4 | 1 | 1 | 1.57 |
| i03.2.6.4.7_10 | 21 | 23 | 24 | 5 | 5 | 1 | 1 | 4.74 |
| i03.2.9.2.0_05 | 13 | 17 | 17 | 4 | 4 | 1 | 1 | 10.94 |
| i03.2.9.2.0_10 | 29 | 29 | 29 | 3 | 3 | 1 | 1 | 18.19 |
| i03.2.9.2.3_05 | 15 | 15 | 15 | 3 | 3 | 1 | 1 | 17.56 |
| i03.2.9.2.3_10 | 40 | 40 | 40 | 3 | 3 | 1 | 1 | 15.11 |
| i03.2.9.2.6_05 | 18 | 18 | 18 | 4 | 4 | 1 | 1 | 13.48 |
| i03.2.9.2.6_10 | 41 | 41 | 41 | 3 | 3 | 1 | 1 | 21.54 |
| i03.2.9.2.9_05 | 8 | 12 | 12 | 6 | 6 | 1 | 1 | 843.46 |
| i03.2.9.2.9_10 | 66 | 66 | 66 | 3 | 3 | 1 | 1 | 811.05 |
| i03.2.9.3.2_05 | 14 | 14 | 15 | 4 | 4 | 1 | 1 | 12.71 |
| i03.2.9.3.2_10 | 38 | 38 | 38 | 4 | 4 | 1 | 1 | 15.34 |
| i03.2.9.3.5_05 | 10 | 13 | 13 | 5 | 6 | 1 | 1 | 63.63 |
| i03.2.9.3.5_10 | 21 | 29 | 30 | 6 | 7 | 1 | 1 | 3990.83 |
| i03.2.9.3.8_05 | 10 | 13 | 13 | 4 | 4 | 1 | 1 | 9.73 |
| i03.2.9.3.8_10 | 14 | 19 | 19 | 6 | 7 | 1 | 1 | 320.76 |
| i03.2.9.4.1_05 | 11 | 13 | 13 | 4 | 4 | 1 | 1 | 10.13 |
| i03.2.9.4.1_10 | 20 | 26 | 27 | 6 | 6 | 1 | 1 | 27.04 |
| i03.2.9.4.4_05 | 9 | 11 | 12 | 4 | 4 | 1 | 1 | 12.28 |
| i03.2.9.4.4_10 | 21 | 27 | 28 | 4 | 4 | 1 | 1 | 11.1 |
| i03.2.9.4.7_05 | 11 | 14 | 15 | 4 | 4 | 1 | 1 | 8.92 |
| i03.2.9.4.7_10 | 22 | 29 | 29 | 4 | 4 | 1 | 1 | 8.83 |
| i03.3.2.2.0_05 | 14 | 14 | 14 | 4 | 4 | 1 | 1 | 160.28 |
| i03.3.2.2.0_10 | 21 | 27 | 27 | 5 | 5 | 1 | 1 | 189.07 |
| i03.3.2.2.3_05 | 10 | 12 | 13 | 4 | 4 | 1 | 1 | 181.87 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i03.3.2.2.3_10 | 21 | 28 | 30 | 4 | 4 | 1 | 1 | 178.95 |
| i03.3.2.2.6_05 | 15 | 15 | 15 | 4 | 4 | 1 | 1 | 320.69 |
| i03.3.2.2.6_10 | 29 | 29 | 29 | 4 | 4 | 1 | 1 | 318.92 |
| i03.3.2.2.9_05 | 10 | 14 | 14 | 4 | 4 | 1 | 1 | 265.25 |
| i03.3.2.2.9_10 | 24 | 31 | 31 | 4 | 4 | 1 | 1 | 250.75 |
| i03.3.2.3.2_05 | 16 | 16 | 16 | 4 | 4 | 1 | 1 | 197.85 |
| i03.3.2.3.2_10 | 32 | 32 | 32 | 6 | 6 | 1 | 1 | 329.37 |
| i03.3.2.3.5_05 | 13 | 13 | 13 | 4 | 4 | 1 | 1 | 309.01 |
| i03.3.2.3.5_10 | 18 | 25 | 26 | 5 | 5 | 1 | 1 | time limit |
| i03.3.2.3.8_05 | 10 | 16 | 15 | 13 | 10 | 1 | 1 | time limit |
| i03.3.2.3.8_10 | 21 | 32 | 32 | 12 | 10 | 1 | 3 | time limit |
| i03.3.2.4.1_05 | 9 | 12 | 12 | 4 | 4 | 1 | 1 | 228.16 |
| i03.3.2.4.1_10 | 20 | 28 | 27 | 4 | 4 | 1 | 3 | 231.28 |
| i03.3.2.4.4_05 | 11 | 15 | 16 | 4 | 4 | 1 | 1 | 276.52 |
| i03.3.2.4.4_10 | 21 | 26 | 26 | 4 | 4 | 1 | 3 | 276.63 |
| i03.3.2.4.7_05 | 10 | 12 | 12 | 4 | 4 | 1 | 1 | 141.14 |
| i03.3.2.4.7_10 | 22 | 30 | 32 | 4 | 4 | 1 | 1 | 152.12 |
| i03.3.5.2.0_05 | 10 | 13 | 14 | 6 | 6 | 1 | 1 | 6052.09 |
| i03.3.5.2.0_10 | 15 | 23 | 24 | 12 | 10 | 1 | 1 | time limit |
| i03.3.5.2.3_05 | 15 | 15 | 15 | 4 | 4 | 1 | 1 | 3728.29 |
| i03.3.5.2.3_10 | 23 | 29 | 30 | 4 | 4 | 1 | 1 | 3756.86 |
| i03.3.5.2.6_05 | 17 | 17 | 17 | 4 | 4 | 1 | 1 | 3745.34 |
| i03.3.5.2.6_10 | 39 | 39 | 39 | 4 | 4 | 1 | 1 | 3736.83 |
| i03.3.5.2.9_05 | 16 | 16 | 17 | 6 | 6 | 2 | 1 | time limit |
| i03.3.5.2.9_10 | 31 | 31 | 31 | 4 | 4 | 1 | 1 | time limit |
| i03.3.5.3.2_05 | 18 | 18 | 18 | 4 | 4 | 1 | 1 | time limit |
| i03.3.5.3.2_10 | 30 | 30 | 30 | 5 | 5 | 1 | 1 | time limit |
| i03.3.5.3.5_05 | 9 | 14 | 15 | 4 | 4 | 1 | 1 | 4886.69 |
| i03.3.5.3.5_10 | 19 | 26 | 27 | 4 | 4 | 1 | 1 | 5042.67 |
| i03.3.5.3.8_05 | 13 | 13 | 13 | 4 | 4 | 1 | 1 | time limit |
| i03.3.5.3.8_10 | 21 | 28 | 29 | 4 | 4 | 1 | 1 | time limit |
| i03.3.5.4.1_05 | 9 | 12 | 12 | 4 | 4 | 1 | 1 | 3553.15 |
| i03.3.5.4.1_10 | 19 | 24 | 25 | 4 | 4 | 1 | 1 | 3582.94 |
| i03.3.5.4.4_05 | 10 | 13 | 15 | 8 | 8 | 1 | 2 | time limit |
| i03.3.5.4.4_10 | 17 | 22 | 23 | 4 | 4 | 1 | 7 | 5203.55 |
| i03.3.5.4.7_05 | 9 | 13 | 13 | 4 | 4 | 1 | 5 | 3311.82 |
| i03.3.5.4.7_10 | 20 | 24 | 24 | 4 | 4 | 1 | 1 | 3042.6 |
| i03.3.8.2.0_05 | 9 | 15 | 16 | 7 | 7 | 1 | 1 | time limit |
| i03.3.8.2.0_10 | 34 | 34 | 34 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.2.3_05 | 20 | 20 | 20 | 3 | 3 | 1 | 1 | time limit |
| i03.3.8.2.3_10 | 33 | 33 | 34 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.2.6_05 | 15 | 15 | 15 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.2.6_10 | 26 | 26 | 26 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.2.9_05 | 29 | 29 | 29 | 3 | 3 | 1 | 1 | time limit |
| i03.3.8.2.9_10 | 36 | 36 | 36 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.3.2_05 | 19 | 19 | 19 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.3.2_10 | 25 | 25 | 25 | 12 | 10 | 1 | 1 | time limit |
| i03.3.8.3.5_05 | 13 | 16 | 16 | 4 | 4 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------------------|------------------------------|-------------------|-------------------|------------|
| i03.3.8.3.5_10 | 20 | 26 | 27 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.3.8_05 | 11 | 14 | 14 | 16 | 12 | 1 | 1 | time limit |
| i03.3.8.3.8_10 | 46 | 46 | 46 | 3 | 3 | 1 | 1 | time limit |
| i03.3.8.4.1_05 | 10 | 13 | 14 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.4.1_10 | 21 | 29 | 31 | 6 | 6 | 1 | 1 | time limit |
| i03.3.8.4.4_05 | 10 | 13 | 13 | 7 | 7 | 1 | 1 | time limit |
| i03.3.8.4.4_10 | 16 | 21 | 23 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.4.7_05 | 9 | 11 | 11 | 4 | 4 | 1 | 1 | time limit |
| i03.3.8.4.7_10 | 24 | 31 | 31 | 3 | 3 | 1 | 1 | time limit |
| i03.4.1.2.0_05 | 7 | 11 | 11 | 15 | 11 | 1 | 1 | time limit |
| i03.4.1.2.0_10 | 27 | 27 | 27 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.2.3_05 | 14 | 14 | 14 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.2.3_10 | 23 | 25 | 25 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.2.6_05 | 24 | 24 | 24 | 3 | 3 | 1 | 1 | time limit |
| i03.4.1.2.6_10 | 25 | 26 | 28 | 6 | 6 | 1 | 1 | time limit |
| i03.4.1.2.9_05 | 16 | 16 | 16 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.2.9_10 | 33 | 33 | 33 | 15 | 11 | 1 | 1 | time limit |
| i03.4.1.3.2_05 | 18 | 18 | 18 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.3.2_10 | 29 | 29 | 29 | 7 | 7 | 1 | 1 | time limit |
| i03.4.1.3.5_05 | 11 | 14 | 15 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.3.5_10 | 18 | 24 | 26 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.3.8_05 | 15 | 15 | 15 | 14 | 11 | 1 | 1 | time limit |
| i03.4.1.3.8_10 | 21 | 30 | 30 | 5 | 5 | 1 | 1 | time limit |
| i03.4.1.4.1_05 | 10 | 14 | 14 | 19 | 14 | 1 | 1 | time limit |
| i03.4.1.4.1_10 | 19 | 24 | 26 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.4.4_05 | 9 | 12 | 13 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.4.4_10 | 40 | 40 | 40 | 3 | 3 | 1 | 1 | time limit |
| i03.4.1.4.7_05 | 11 | 15 | 15 | 4 | 4 | 1 | 1 | time limit |
| i03.4.1.4.7_10 | 19 | 22 | 22 | 4 | 4 | 1 | 4 | time limit |
| i03.4.4.2.0_05 | 13 | 14 | 15 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.2.0_10 | 14 | 22 | 23 | 7 | 7 | 1 | 1 | time limit |
| i03.4.4.2.3_05 | 18 | 18 | 18 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.2.3_10 | 27 | 27 | 27 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.2.6_05 | 9 | 13 | 13 | 10 | 10 | 1 | 1 | time limit |
| i03.4.4.2.6_10 | 51 | 51 | 51 | 3 | 3 | 1 | 1 | time limit |
| i03.4.4.2.9_05 | 25 | 25 | 25 | 3 | 3 | 1 | 1 | time limit |
| i03.4.4.2.9_10 | 63 | 63 | 63 | 3 | 3 | 1 | 1 | time limit |
| i03.4.4.3.2_05 | 14 | 14 | 14 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.3.2_10 | 30 | 30 | 30 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.3.5_05 | 17 | 18 | 17 | 3 | 3 | 3 | 3 | time limit |
| i03.4.4.3.5_10 | 18 | 25 | 25 | 7 | 7 | 1 | 1 | time limit |
| i03.4.4.3.8_05 | 16 | 16 | 16 | 3 | 3 | 1 | 1 | time limit |
| i03.4.4.3.8_10 | 23 | 23 | 27 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.4.1_05 | 11 | 13 | 13 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.4.1_10 | 21 | 28 | 29 | 6 | 6 | 1 | 7 | time limit |
| i03.4.4.4.4_05 | 9 | 11 | 11 | 4 | 4 | 1 | 1 | time limit |
| i03.4.4.4.4_10 | 31 | 31 | 31 | 3 | 3 | 1 | 1 | time limit |
| i03.4.4.4.7_05 | 14 | 18 | 18 | 3 | 3 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i03.4.4.4.7_10 | 18 | 23 | 24 | 16 | 12 | 1 | 1 | time limit |
| i03.4.7.2.0_05 | 11 | 14 | 15 | 6 | 6 | 1 | 1 | time limit |
| i03.4.7.2.0_10 | 16 | 25 | 26 | 12 | 10 | 1 | 1 | time limit |
| i03.4.7.2.3_05 | 17 | 17 | 17 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.2.3_10 | 29 | 29 | 29 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.2.6_05 | 11 | 11 | 11 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.2.6_10 | 34 | 34 | 34 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.2.9_05 | 11 | 15 | 15 | 22 | 16 | 1 | 1 | time limit |
| i03.4.7.2.9_10 | 31 | 31 | 31 | 7 | 7 | 1 | 1 | time limit |
| i03.4.7.3.2_05 | 15 | 15 | 15 | 8 | 8 | 1 | 1 | time limit |
| i03.4.7.3.2_10 | 60 | 60 | 60 | 3 | 3 | 1 | 1 | time limit |
| i03.4.7.3.5_05 | 24 | 24 | 24 | 3 | 3 | 1 | 1 | time limit |
| i03.4.7.3.5_10 | 20 | 28 | 28 | 5 | 5 | 1 | 1 | time limit |
| i03.4.7.3.8_05 | 13 | 13 | 13 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.3.8_10 | 16 | 27 | 28 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.4.1_05 | 16 | 16 | 16 | 3 | 3 | 1 | 1 | time limit |
| i03.4.7.4.1_10 | 17 | 24 | 24 | 6 | 6 | 1 | 1 | time limit |
| i03.4.7.4.4_05 | 8 | 11 | 11 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.4.4_10 | 18 | 21 | 21 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.4.7_05 | 11 | 13 | 14 | 4 | 4 | 1 | 1 | time limit |
| i03.4.7.4.7_10 | 17 | 25 | 26 | 4 | 4 | 1 | 1 | time limit |
| i04.2.0.2.0_05 | 24 | 24 | 24 | 6 | 6 | 1 | 1 | 0.46 |
| i04.2.0.2.0_10 | 50 | 50 | 50 | 7 | 7 | 1 | 1 | 1.02 |
| i04.2.0.2.3_05 | 24 | 24 | 24 | 6 | 6 | 1 | 1 | 0.78 |
| i04.2.0.2.3_10 | 36 | 36 | 37 | 7 | 7 | 1 | 1 | 0.68 |
| i04.2.0.2.6_05 | 25 | 25 | 27 | 6 | 6 | 1 | 1 | 0.66 |
| i04.2.0.2.6_10 | 33 | 33 | 32 | 7 | 7 | 1 | 1 | 0.6 |
| i04.2.0.2.9_05 | 33 | 33 | 33 | 6 | 6 | 1 | 1 | 0.7 |
| i04.2.0.2.9_10 | 22 | 33 | 33 | 7 | 7 | 1 | 1 | 0.73 |
| i04.2.0.3.2_05 | 39 | 39 | 39 | 6 | 6 | 1 | 1 | 0.77 |
| i04.2.0.3.2_10 | 44 | 44 | 44 | 6 | 6 | 1 | 1 | 0.54 |
| i04.2.0.3.5_05 | 12 | 17 | 17 | 7 | 7 | 1 | 1 | 0.63 |
| i04.2.0.3.5_10 | 25 | 33 | 34 | 7 | 7 | 1 | 1 | 0.55 |
| i04.2.0.3.8_05 | 17 | 23 | 24 | 6 | 6 | 1 | 1 | 1.66 |
| i04.2.0.3.8_10 | 27 | 35 | 36 | 6 | 6 | 1 | 2 | 0.49 |
| i04.2.0.4.1_05 | 11 | 15 | 16 | 7 | 7 | 1 | 1 | 0.62 |
| i04.2.0.4.1_10 | 21 | 30 | 31 | 7 | 7 | 1 | 1 | 0.45 |
| i04.2.0.4.4_05 | 15 | 20 | 21 | 7 | 7 | 1 | 4 | 1.04 |
| i04.2.0.4.4_10 | 21 | 30 | 30 | 6 | 6 | 1 | 1 | 0.5 |
| i04.2.0.4.7_05 | 12 | 16 | 17 | 7 | 7 | 1 | 1 | 0.8 |
| i04.2.0.4.7_10 | 40 | 55 | 58 | 6 | 6 | 1 | 1 | 1.91 |
| i04.2.3.2.0_05 | 19 | 19 | 19 | 6 | 6 | 1 | 1 | 0.86 |
| i04.2.3.2.0_10 | 47 | 47 | 50 | 9 | 9 | 1 | 1 | 2.79 |
| i04.2.3.2.3_05 | 16 | 18 | 18 | 7 | 7 | 1 | 1 | 0.97 |
| i04.2.3.2.3_10 | 36 | 47 | 45 | 7 | 7 | 1 | 1 | 1.06 |
| i04.2.3.2.6_05 | 13 | 20 | 20 | 7 | 7 | 1 | 1 | 0.84 |
| i04.2.3.2.6_10 | 25 | 33 | 34 | 7 | 7 | 1 | 1 | 1.07 |
| i04.2.3.2.9_05 | 36 | 36 | 36 | 6 | 6 | 1 | 1 | 0.99 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|-----------|
| i04.2.3.2.9_10 | 44 | 44 | 44 | 6 | 6 | 1 | 1 | 0.71 |
| i04.2.3.3.2_05 | 38 | 38 | 32 | 5 | 5 | 1 | 1 | 1.31 |
| i04.2.3.3.2_10 | 41 | 41 | 41 | 6 | 6 | 1 | 1 | 1.18 |
| i04.2.3.3.5_05 | 24 | 24 | 24 | 6 | 6 | 1 | 1 | 1.07 |
| i04.2.3.3.5_10 | 39 | 39 | 40 | 8 | 8 | 1 | 1 | 3.73 |
| i04.2.3.3.8_05 | 17 | 21 | 23 | 6 | 6 | 1 | 1 | 0.77 |
| i04.2.3.3.8_10 | 20 | 26 | 26 | 7 | 7 | 1 | 1 | 1.17 |
| i04.2.3.4.1_05 | 11 | 15 | 15 | 7 | 7 | 1 | 1 | 1.96 |
| i04.2.3.4.1_10 | 25 | 33 | 35 | 7 | 7 | 1 | 9 | 1.08 |
| i04.2.3.4.4_05 | 14 | 18 | 19 | 10 | 10 | 1 | 1 | 3.77 |
| i04.2.3.4.4_10 | 35 | 36 | 36 | 6 | 6 | 1 | 1 | 1.14 |
| i04.2.3.4.7_05 | 14 | 19 | 19 | 6 | 6 | 1 | 1 | 0.94 |
| i04.2.3.4.7_10 | 29 | 38 | 39 | 6 | 6 | 1 | 3 | 0.87 |
| i04.2.6.2.0_05 | 23 | 23 | 23 | 9 | 10 | 1 | 1 | 814.78 |
| i04.2.6.2.0_10 | 24 | 29 | 30 | 7 | 7 | 1 | 1 | 3.18 |
| i04.2.6.2.3_05 | 16 | 18 | 18 | 8 | 8 | 1 | 1 | 5.12 |
| i04.2.6.2.3_10 | 56 | 56 | 56 | 4 | 4 | 1 | 1 | 2.89 |
| i04.2.6.2.6_05 | 13 | 16 | 16 | 9 | 9 | 1 | 1 | 5.7 |
| i04.2.6.2.6_10 | 49 | 49 | 49 | 6 | 6 | 1 | 1 | 2.76 |
| i04.2.6.2.9_05 | 13 | 17 | 17 | 9 | 9 | 1 | 1 | 5.42 |
| i04.2.6.2.9_10 | 25 | 34 | 34 | 11 | 11 | 1 | 1 | 11.48 |
| i04.2.6.3.2_05 | 30 | 30 | 30 | 6 | 6 | 1 | 1 | 2.67 |
| i04.2.6.3.2_10 | 25 | 34 | 35 | 7 | 7 | 1 | 1 | 2.09 |
| i04.2.6.3.5_05 | 11 | 14 | 23 | 10 | 10 | 1 | 1 | 2087.48 |
| i04.2.6.3.5_10 | 35 | 35 | 35 | 6 | 6 | 1 | 1 | 3.16 |
| i04.2.6.3.8_05 | 15 | 19 | 20 | 6 | 6 | 1 | 1 | 1.72 |
| i04.2.6.3.8_10 | 46 | 47 | 46 | 10 | 10 | 3 | 3 | 15.18 |
| i04.2.6.4.1_05 | 12 | 15 | 16 | 7 | 7 | 1 | 1 | 2.91 |
| i04.2.6.4.1_10 | 22 | 30 | 32 | 9 | 9 | 1 | 1 | 7.83 |
| i04.2.6.4.4_05 | 13 | 17 | 17 | 10 | 10 | 1 | 1 | 9.94 |
| i04.2.6.4.4_10 | 22 | 28 | 29 | 7 | 7 | 1 | 1 | 1.98 |
| i04.2.6.4.7_05 | 15 | 21 | 21 | 6 | 6 | 1 | 25 | 1.98 |
| i04.2.6.4.7_10 | 26 | 33 | 31 | 8 | 8 | 1 | 1 | 6.94 |
| i04.2.9.2.0_05 | 20 | 20 | 20 | 7 | 7 | 1 | 1 | 11.42 |
| i04.2.9.2.0_10 | 21 | 30 | 31 | 8 | 8 | 1 | 1 | 28.55 |
| i04.2.9.2.3_05 | 12 | 17 | 17 | 7 | 7 | 1 | 1 | 12.66 |
| i04.2.9.2.3_10 | 27 | 37 | 38 | 7 | 7 | 1 | 1 | 14.28 |
| i04.2.9.2.6_05 | 14 | 19 | 20 | 7 | 7 | 1 | 1 | 13.73 |
| i04.2.9.2.6_10 | 36 | 36 | 36 | 7 | 7 | 1 | 1 | 20.81 |
| i04.2.9.2.9_05 | 13 | 16 | 16 | 8 | 8 | 1 | 1 | 833.45 |
| i04.2.9.2.9_10 | 29 | 38 | 39 | 9 | 9 | 1 | 1 | 828.9 |
| i04.2.9.3.2_05 | 13 | 17 | 18 | 7 | 7 | 1 | 1 | 13.44 |
| i04.2.9.3.2_10 | 27 | 36 | 37 | 7 | 7 | 1 | 1 | 13.17 |
| i04.2.9.3.5_05 | 24 | 24 | 24 | 7 | 7 | 1 | 1 | 17.61 |
| i04.2.9.3.5_10 | 25 | 33 | 35 | 7 | 7 | 1 | 1 | 14.5 |
| i04.2.9.3.8_05 | 13 | 17 | 18 | 7 | 7 | 1 | 1 | 10.21 |
| i04.2.9.3.8_10 | 25 | 33 | 34 | 8 | 8 | 1 | 1 | 22.27 |
| i04.2.9.4.1_05 | 14 | 17 | 16 | 6 | 6 | 1 | 1 | 11.17 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i04.2.9.4.1_10 | 28 | 28 | 29 | 10 | 10 | 1 | 1 | 132.03 |
| i04.2.9.4.4_05 | 13 | 17 | 17 | 7 | 7 | 1 | 1 | 12.91 |
| i04.2.9.4.4_10 | 24 | 31 | 31 | 7 | 7 | 1 | 1 | 12.37 |
| i04.2.9.4.7_05 | 12 | 15 | 16 | 7 | 7 | 1 | 5 | 11.01 |
| i04.2.9.4.7_10 | 25 | 33 | 35 | 7 | 7 | 1 | 3 | 10.55 |
| i04.3.2.2.0_05 | 15 | 19 | 20 | 8 | 9 | 1 | 1 | 687.14 |
| i04.3.2.2.0_10 | 32 | 36 | 38 | 10 | 10 | 1 | 1 | 3399.16 |
| i04.3.2.2.3_05 | 14 | 19 | 20 | 17 | 15 | 1 | 1 | time limit |
| i04.3.2.2.3_10 | 38 | 38 | 38 | 8 | 8 | 1 | 1 | 200.74 |
| i04.3.2.2.6_05 | 24 | 24 | 24 | 6 | 6 | 1 | 1 | 356.41 |
| i04.3.2.2.6_10 | 35 | 52 | 53 | 6 | 6 | 1 | 1 | 335.98 |
| i04.3.2.2.9_05 | 32 | 32 | 32 | 6 | 6 | 1 | 1 | 280.78 |
| i04.3.2.2.9_10 | 33 | 33 | 33 | 13 | 13 | 1 | 1 | 1902.63 |
| i04.3.2.3.2_05 | 14 | 20 | 21 | 18 | 16 | 1 | 1 | time limit |
| i04.3.2.3.2_10 | 37 | 38 | 41 | 7 | 7 | 1 | 1 | 202.23 |
| i04.3.2.3.5_05 | 13 | 17 | 18 | 9 | 9 | 1 | 1 | 442.49 |
| i04.3.2.3.5_10 | 24 | 34 | 36 | 7 | 7 | 1 | 1 | 316.43 |
| i04.3.2.3.8_05 | 12 | 18 | 19 | 6 | 6 | 1 | 1 | time limit |
| i04.3.2.3.8_10 | 18 | 23 | 24 | 12 | 12 | 1 | 1 | time limit |
| i04.3.2.4.1_05 | 10 | 14 | 14 | 7 | 7 | 1 | 1 | 229.88 |
| i04.3.2.4.1_10 | 25 | 34 | 37 | 11 | 11 | 1 | 1 | time limit |
| i04.3.2.4.4_05 | 17 | 22 | 20 | 5 | 5 | 1 | 3 | 333.53 |
| i04.3.2.4.4_10 | 26 | 38 | 39 | 6 | 6 | 1 | 1 | 285.02 |
| i04.3.2.4.7_05 | 13 | 17 | 18 | 7 | 7 | 1 | 1 | 147.94 |
| i04.3.2.4.7_10 | 22 | 31 | 32 | 7 | 7 | 1 | 1 | 144.18 |
| i04.3.5.2.0_05 | 15 | 19 | 19 | 6 | 6 | 1 | 1 | 5188.98 |
| i04.3.5.2.0_10 | 27 | 35 | 36 | 18 | 16 | 1 | 1 | time limit |
| i04.3.5.2.3_05 | 12 | 17 | 17 | 19 | 18 | 1 | 1 | time limit |
| i04.3.5.2.3_10 | 34 | 34 | 34 | 6 | 6 | 1 | 1 | 3724.21 |
| i04.3.5.2.6_05 | 18 | 18 | 19 | 7 | 7 | 3 | 1 | 3596.22 |
| i04.3.5.2.6_10 | 28 | 38 | 41 | 19 | 17 | 1 | 2 | time limit |
| i04.3.5.2.9_05 | 14 | 19 | 19 | 7 | 7 | 1 | 1 | time limit |
| i04.3.5.2.9_10 | 24 | 31 | 31 | 13 | 13 | 1 | 1 | time limit |
| i04.3.5.3.2_05 | 14 | 17 | 18 | 7 | 7 | 1 | 2 | time limit |
| i04.3.5.3.2_10 | 34 | 34 | 34 | 7 | 7 | 1 | 1 | time limit |
| i04.3.5.3.5_05 | 17 | 17 | 17 | 9 | 9 | 1 | 1 | time limit |
| i04.3.5.3.5_10 | 27 | 38 | 47 | 69 | 39 | 1 | 1 | time limit |
| i04.3.5.3.8_05 | 12 | 15 | 16 | 10 | 10 | 1 | 12 | time limit |
| i04.3.5.3.8_10 | 22 | 29 | 29 | 8 | 8 | 1 | 1 | time limit |
| i04.3.5.4.1_05 | 17 | 18 | 19 | 6 | 6 | 1 | 1 | 3703.7 |
| i04.3.5.4.1_10 | 28 | 33 | 33 | 6 | 6 | 1 | 1 | 3775.55 |
| i04.3.5.4.4_05 | 11 | 14 | 15 | 7 | 7 | 1 | 1 | 5490.41 |
| i04.3.5.4.4_10 | 22 | 30 | 31 | 7 | 7 | 1 | 1 | 5245.52 |
| i04.3.5.4.7_05 | 17 | 20 | 21 | 6 | 6 | 1 | 1 | 3241.22 |
| i04.3.5.4.7_10 | 20 | 26 | 26 | 7 | 7 | 1 | 1 | 3024.1 |
| i04.3.8.2.0_05 | 17 | 17 | 17 | 8 | 8 | 1 | 1 | time limit |
| i04.3.8.2.0_10 | 36 | 36 | 36 | 6 | 6 | 1 | 1 | time limit |
| i04.3.8.2.3_05 | 10 | 14 | 18 | 19 | 17 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i04.3.8.2.3_10 | 39 | 39 | 39 | 10 | 10 | 1 | 1 | time limit |
| i04.3.8.2.6_05 | 14 | 20 | 21 | 27 | 19 | 1 | 2 | time limit |
| i04.3.8.2.6_10 | 46 | 46 | 46 | 6 | 6 | 1 | 1 | time limit |
| i04.3.8.2.9_05 | 42 | 42 | 42 | 4 | 4 | 1 | 1 | time limit |
| i04.3.8.2.9_10 | 54 | 54 | 54 | 6 | 6 | 1 | 1 | time limit |
| i04.3.8.3.2_05 | 36 | 36 | 36 | 6 | 6 | 1 | 1 | time limit |
| i04.3.8.3.2_10 | 71 | 71 | 71 | 6 | 6 | 1 | 1 | time limit |
| i04.3.8.3.5_05 | 11 | 14 | 14 | 7 | 7 | 1 | 1 | time limit |
| i04.3.8.3.5_10 | 38 | 38 | 38 | 12 | 12 | 1 | 1 | time limit |
| i04.3.8.3.8_05 | 11 | 15 | 17 | 7 | 7 | 1 | 1 | time limit |
| i04.3.8.3.8_10 | 55 | 56 | 55 | 4 | 4 | 3 | 3 | time limit |
| i04.3.8.4.1_05 | 10 | 14 | 15 | 15 | 13 | 1 | 4 | time limit |
| i04.3.8.4.1_10 | 26 | 36 | 37 | 9 | 9 | 1 | 8 | time limit |
| i04.3.8.4.4_05 | 12 | 17 | 17 | 18 | 16 | 1 | 1 | time limit |
| i04.3.8.4.4_10 | 20 | 26 | 27 | 12 | 12 | 1 | 1 | time limit |
| i04.3.8.4.7_05 | 8 | 10 | 14 | 24 | 18 | 1 | 1 | time limit |
| i04.3.8.4.7_10 | 23 | 32 | 32 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.2.0_05 | 13 | 19 | 19 | 11 | 11 | 1 | 1 | time limit |
| i04.4.1.2.0_10 | 31 | 40 | 40 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.2.3_05 | 15 | 15 | 16 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.2.3_10 | 50 | 50 | 50 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.2.6_05 | 19 | 19 | 19 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.2.6_10 | 48 | 48 | 48 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.2.9_05 | 18 | 18 | 18 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.2.9_10 | 26 | 35 | 35 | 30 | 22 | 1 | 1 | time limit |
| i04.4.1.3.2_05 | 18 | 18 | 19 | 12 | 12 | 1 | 1 | time limit |
| i04.4.1.3.2_10 | 40 | 40 | 40 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.3.5_05 | 19 | 19 | 23 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.3.5_10 | 35 | 36 | 36 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.3.8_05 | 18 | 18 | 19 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.3.8_10 | 22 | 31 | 32 | 7 | 7 | 1 | 1 | time limit |
| i04.4.1.4.1_05 | 9 | 13 | 13 | 25 | 18 | 1 | 1 | time limit |
| i04.4.1.4.1_10 | 35 | 37 | 35 | 6 | 6 | 1 | 7 | time limit |
| i04.4.1.4.4_05 | 17 | 19 | 17 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.4.4_10 | 19 | 28 | 29 | 18 | 16 | 1 | 1 | time limit |
| i04.4.1.4.7_05 | 11 | 17 | 17 | 6 | 6 | 1 | 1 | time limit |
| i04.4.1.4.7_10 | 21 | 35 | 37 | 6 | 6 | 1 | 1 | time limit |
| i04.4.4.2.0_05 | 19 | 19 | 19 | 6 | 6 | 1 | 1 | time limit |
| i04.4.4.2.0_10 | 18 | 27 | 27 | 29 | 22 | 1 | 1 | time limit |
| i04.4.4.2.3_05 | 14 | 14 | 14 | 30 | 22 | 1 | 1 | time limit |
| i04.4.4.2.3_10 | 24 | 36 | 37 | 32 | 24 | 1 | 1 | time limit |
| i04.4.4.2.6_05 | 28 | 28 | 28 | 6 | 6 | 1 | 1 | time limit |
| i04.4.4.2.6_10 | 25 | 35 | 35 | 104 | 59 | 1 | 1 | time limit |
| i04.4.4.2.9_05 | 13 | 18 | 21 | 159 | 87 | 1 | 1 | time limit |
| i04.4.4.2.9_10 | 43 | 43 | 43 | 7 | 7 | 1 | 1 | time limit |
| i04.4.4.3.2_05 | 28 | 28 | 28 | 11 | 11 | 1 | 1 | time limit |
| i04.4.4.3.2_10 | 41 | 41 | 41 | 8 | 8 | 1 | 1 | time limit |
| i04.4.4.3.5_10 | 22 | 29 | 29 | 12 | 12 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i04.4.4.3.8_05 | 24 | 24 | 24 | 21 | 16 | 1 | 1 | time limit |
| i04.4.4.3.8_10 | 23 | 29 | 32 | 7 | 7 | 1 | 1 | time limit |
| i04.4.4.4.1_05 | 18 | 18 | 18 | 6 | 6 | 1 | 1 | time limit |
| i04.4.4.4.1_10 | 25 | 35 | 36 | 24 | 18 | 1 | 1 | time limit |
| i04.4.4.4.4_05 | 15 | 15 | 15 | 6 | 6 | 1 | 1 | time limit |
| i04.4.4.4.4_10 | 26 | 34 | 35 | 13 | 13 | 1 | 3 | time limit |
| i04.4.4.4.7_05 | 13 | 18 | 18 | 9 | 9 | 1 | 1 | time limit |
| i04.4.4.4.7_10 | 27 | 36 | 36 | 11 | 11 | 1 | 1 | time limit |
| i04.4.7.2.0_05 | 0 | 16 | 16 | 148 | 88 | 0 | 1 | time limit |
| i04.4.7.2.0_10 | 17 | 26 | 27 | 34 | 25 | 1 | 1 | time limit |
| i04.4.7.2.3_05 | 14 | 17 | 18 | 11 | 11 | 1 | 1 | time limit |
| i04.4.7.2.3_10 | 20 | 30 | 31 | 22 | 19 | 1 | 1 | time limit |
| i04.4.7.2.6_05 | 23 | 23 | 23 | 6 | 6 | 1 | 1 | time limit |
| i04.4.7.2.6_10 | 33 | 33 | 33 | 7 | 7 | 1 | 1 | time limit |
| i04.4.7.2.9_05 | 17 | 17 | 17 | 10 | 10 | 1 | 1 | time limit |
| i04.4.7.2.9_10 | 28 | 50 | 54 | 6 | 6 | 1 | 1 | time limit |
| i04.4.7.3.2_05 | 16 | 16 | 17 | 7 | 7 | 1 | 1 | time limit |
| i04.4.7.3.2_10 | 34 | 34 | 34 | 8 | 8 | 1 | 1 | time limit |
| i04.4.7.3.5_05 | 16 | 22 | 22 | 6 | 6 | 1 | 1 | time limit |
| i04.4.7.3.5_10 | 42 | 42 | 42 | 6 | 6 | 1 | 1 | time limit |
| i04.4.7.3.8_05 | 12 | 17 | 17 | 13 | 12 | 1 | 1 | time limit |
| i04.4.7.3.8_10 | 18 | 26 | 26 | 27 | 20 | 1 | 1 | time limit |
| i04.4.7.4.1_05 | 11 | 16 | 15 | 37 | 25 | 1 | 1 | time limit |
| i04.4.7.4.1_10 | 34 | 34 | 34 | 30 | 22 | 1 | 1 | time limit |
| i04.4.7.4.4_05 | 11 | 14 | 15 | 7 | 7 | 1 | 1 | time limit |
| i04.4.7.4.4_10 | 18 | 25 | 25 | 34 | 24 | 1 | 1 | time limit |
| i04.4.7.4.7_05 | 16 | 21 | 21 | 12 | 12 | 1 | 4 | time limit |
| i04.4.7.4.7_10 | 23 | 30 | 33 | 6 | 6 | 1 | 1 | time limit |
| i05.2.0.2.0_05 | 24 | 24 | 24 | 10 | 10 | 1 | 1 | 0.55 |
| i05.2.0.2.0_10 | 59 | 59 | 59 | 10 | 10 | 1 | 1 | 1.42 |
| i05.2.0.2.3_05 | 25 | 25 | 26 | 10 | 10 | 1 | 1 | 0.81 |
| i05.2.0.2.3_10 | 47 | 47 | 47 | 10 | 10 | 1 | 1 | 0.74 |
| i05.2.0.2.6_05 | 31 | 31 | 32 | 9 | 9 | 1 | 1 | 0.85 |
| i05.2.0.2.6_10 | 46 | 46 | 46 | 10 | 10 | 1 | 1 | 0.69 |
| i05.2.0.2.9_05 | 33 | 33 | 33 | 9 | 9 | 1 | 1 | 0.74 |
| i05.2.0.2.9_10 | 50 | 50 | 50 | 10 | 10 | 1 | 1 | 1.9 |
| i05.2.0.3.2_05 | 26 | 26 | 26 | 9 | 9 | 1 | 1 | 0.64 |
| i05.2.0.3.2_10 | 40 | 40 | 40 | 11 | 11 | 1 | 1 | 0.81 |
| i05.2.0.3.5_05 | 15 | 19 | 19 | 10 | 10 | 1 | 1 | 0.94 |
| i05.2.0.3.5_10 | 25 | 37 | 40 | 10 | 10 | 1 | 1 | 0.92 |
| i05.2.0.3.8_05 | 15 | 22 | 23 | 11 | 11 | 1 | 3 | 1.12 |
| i05.2.0.3.8_10 | 44 | 44 | 44 | 10 | 10 | 1 | 1 | 0.8 |
| i05.2.0.4.1_05 | 17 | 20 | 21 | 8 | 8 | 1 | 1 | 0.88 |
| i05.2.0.4.1_10 | 34 | 46 | 48 | 10 | 10 | 1 | 2 | 1.14 |
| i05.2.0.4.4_05 | 12 | 18 | 19 | 10 | 10 | 1 | 1 | 0.67 |
| i05.2.0.4.4_10 | 29 | 40 | 43 | 10 | 10 | 1 | 1 | 0.6 |
| i05.2.0.4.7_05 | 15 | 23 | 22 | 9 | 9 | 1 | 1 | 0.89 |
| i05.2.0.4.7_10 | 25 | 34 | 36 | 10 | 10 | 1 | 1 | 0.8 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|-----------|
| i05_2.3.2.0_05 | 24 | 24 | 26 | 10 | 10 | 1 | 1 | 1.1 |
| i05_2.3.2.0_10 | 53 | 53 | 58 | 12 | 12 | 1 | 1 | 2.46 |
| i05_2.3.2.3_05 | 33 | 33 | 33 | 8 | 8 | 1 | 1 | 1.44 |
| i05_2.3.2.3_10 | 61 | 61 | 61 | 10 | 10 | 1 | 1 | 1.37 |
| i05_2.3.2.6_05 | 40 | 40 | 40 | 10 | 10 | 1 | 1 | 1.09 |
| i05_2.3.2.6_10 | 33 | 53 | 55 | 13 | 13 | 1 | 1 | 3.23 |
| i05_2.3.2.9_05 | 30 | 30 | 30 | 10 | 10 | 1 | 1 | 2.25 |
| i05_2.3.2.9_10 | 61 | 61 | 61 | 12 | 12 | 1 | 1 | 9.73 |
| i05_2.3.3.2_05 | 32 | 32 | 32 | 9 | 9 | 1 | 1 | 1.32 |
| i05_2.3.3.2_10 | 34 | 47 | 50 | 14 | 14 | 1 | 1 | 5.22 |
| i05_2.3.3.5_05 | 17 | 25 | 25 | 13 | 13 | 1 | 9 | 6.03 |
| i05_2.3.3.5_10 | 23 | 33 | 34 | 12 | 12 | 1 | 1 | 2.0 |
| i05_2.3.3.8_05 | 14 | 19 | 21 | 14 | 14 | 1 | 1 | 10.89 |
| i05_2.3.3.8_10 | 37 | 52 | 55 | 17 | 17 | 1 | 2 | 10.71 |
| i05_2.3.4.1_05 | 12 | 16 | 16 | 12 | 12 | 1 | 2 | 2.1 |
| i05_2.3.4.1_10 | 26 | 37 | 39 | 10 | 10 | 1 | 1 | 1.48 |
| i05_2.3.4.4_05 | 16 | 23 | 24 | 9 | 9 | 1 | 1 | 1.48 |
| i05_2.3.4.4_10 | 51 | 51 | 51 | 8 | 8 | 1 | 1 | 1.4 |
| i05_2.3.4.7_05 | 14 | 18 | 19 | 10 | 10 | 1 | 1 | 1.38 |
| i05_2.3.4.7_10 | 26 | 34 | 36 | 10 | 10 | 1 | 1 | 1.12 |
| i05_2.6.2.0_05 | 19 | 19 | 20 | 11 | 11 | 1 | 1 | 2.46 |
| i05_2.6.2.0_10 | 31 | 41 | 41 | 10 | 10 | 1 | 1 | 2.1 |
| i05_2.6.2.3_05 | 17 | 24 | 24 | 14 | 14 | 1 | 1 | 10.51 |
| i05_2.6.2.3_10 | 54 | 54 | 56 | 10 | 10 | 1 | 1 | 4.46 |
| i05_2.6.2.6_05 | 36 | 36 | 26 | 10 | 10 | 1 | 1 | 3.14 |
| i05_2.6.2.6_10 | 49 | 49 | 49 | 10 | 10 | 1 | 1 | 2.95 |
| i05_2.6.2.9_10 | 58 | 58 | 58 | 10 | 10 | 1 | 1 | 3.35 |
| i05_2.6.3.2_05 | 22 | 22 | 22 | 10 | 10 | 1 | 1 | 2.58 |
| i05_2.6.3.2_10 | 76 | 76 | 76 | 14 | 14 | 1 | 1 | 15.57 |
| i05_2.6.3.5_05 | 12 | 17 | 18 | 12 | 12 | 1 | 1 | 4.37 |
| i05_2.6.3.5_10 | 30 | 40 | 41 | 11 | 11 | 1 | 1 | 3.45 |
| i05_2.6.3.8_05 | 16 | 21 | 22 | 10 | 10 | 1 | 1 | 3.66 |
| i05_2.6.3.8_10 | 21 | 29 | 29 | 12 | 12 | 1 | 3 | 6.26 |
| i05_2.6.4.1_05 | 14 | 20 | 21 | 10 | 10 | 1 | 4 | 2.59 |
| i05_2.6.4.1_10 | 45 | 46 | 46 | 8 | 8 | 1 | 7 | 5.03 |
| i05_2.6.4.4_05 | 12 | 15 | 16 | 13 | 13 | 1 | 1 | 7.12 |
| i05_2.6.4.4_10 | 32 | 41 | 45 | 10 | 10 | 1 | 1 | 4.04 |
| i05_2.6.4.7_05 | 13 | 17 | 18 | 13 | 13 | 1 | 1 | 5.69 |
| i05_2.6.4.7_10 | 32 | 41 | 41 | 10 | 10 | 1 | 1 | 2.73 |
| i05_2.9.2.0_05 | 18 | 22 | 23 | 12 | 13 | 1 | 1 | 74.19 |
| i05_2.9.2.0_10 | 31 | 48 | 49 | 14 | 18 | 1 | 1 | 3994.57 |
| i05_2.9.2.3_05 | 27 | 27 | 27 | 8 | 8 | 1 | 1 | 19.24 |
| i05_2.9.2.3_10 | 67 | 67 | 67 | 9 | 9 | 1 | 1 | 20.63 |
| i05_2.9.2.6_05 | 27 | 27 | 27 | 10 | 10 | 1 | 1 | 16.67 |
| i05_2.9.2.6_10 | 44 | 44 | 44 | 10 | 10 | 1 | 1 | 19.49 |
| i05_2.9.2.9_05 | 17 | 24 | 24 | 17 | 19 | 1 | 1 | 4983.49 |
| i05_2.9.2.9_10 | 46 | 46 | 46 | 11 | 11 | 2 | 2 | 817.14 |
| i05_2.9.3.2_05 | 18 | 18 | 19 | 12 | 12 | 5 | 5 | 23.49 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i05.2.9.3.2_10 | 47 | 47 | 47 | 14 | 18 | 1 | 1 | 369.7 |
| i05.2.9.3.5_05 | 15 | 20 | 19 | 13 | 14 | 1 | 1 | 3887.41 |
| i05.2.9.3.5_10 | 32 | 42 | 45 | 15 | 16 | 1 | 2 | 2519.34 |
| i05.2.9.3.8_05 | 16 | 20 | 21 | 9 | 9 | 1 | 1 | 14.52 |
| i05.2.9.3.8_10 | 31 | 43 | 45 | 12 | 15 | 1 | 1 | 264.68 |
| i05.2.9.4.1_05 | 16 | 20 | 22 | 10 | 10 | 1 | 3 | 15.38 |
| i05.2.9.4.1_10 | 36 | 47 | 51 | 17 | 18 | 1 | 2 | 1080.04 |
| i05.2.9.4.4_05 | 14 | 19 | 20 | 22 | 32 | 1 | 4 | time limit |
| i05.2.9.4.4_10 | 33 | 43 | 44 | 10 | 10 | 1 | 1 | 15.23 |
| i05.2.9.4.7_05 | 14 | 18 | 18 | 11 | 11 | 1 | 1 | 10.37 |
| i05.2.9.4.7_10 | 25 | 32 | 32 | 10 | 10 | 1 | 1 | 10.61 |
| i05.3.2.2.0_05 | 16 | 22 | 22 | 16 | 17 | 1 | 1 | 1379.43 |
| i05.3.2.2.0_10 | 33 | 43 | 45 | 16 | 16 | 1 | 1 | 1101.96 |
| i05.3.2.2.3_05 | 14 | 19 | 20 | 21 | 22 | 1 | 1 | 3045.54 |
| i05.3.2.2.3_10 | 65 | 65 | 65 | 9 | 9 | 1 | 1 | 261.12 |
| i05.3.2.2.6_05 | 19 | 24 | 26 | 14 | 14 | 1 | 1 | time limit |
| i05.3.2.2.6_10 | 51 | 51 | 52 | 15 | 15 | 1 | 1 | 4556.75 |
| i05.3.2.2.9_05 | 43 | 43 | 43 | 8 | 8 | 1 | 1 | 318.92 |
| i05.3.2.2.9_10 | 39 | 39 | 39 | 15 | 15 | 1 | 1 | 378.87 |
| i05.3.2.3.2_05 | 29 | 29 | 29 | 10 | 10 | 1 | 1 | 246.05 |
| i05.3.2.3.2_10 | 38 | 38 | 38 | 10 | 10 | 1 | 1 | 211.9 |
| i05.3.2.3.5_05 | 13 | 17 | 19 | 11 | 11 | 1 | 1 | 358.43 |
| i05.3.2.3.5_10 | 28 | 47 | 51 | 10 | 10 | 1 | 1 | 325.0 |
| i05.3.2.3.8_05 | 13 | 20 | 22 | 26 | 30 | 1 | 1 | time limit |
| i05.3.2.3.8_10 | 22 | 31 | 34 | 10 | 10 | 1 | 1 | time limit |
| i05.3.2.4.1_05 | 21 | 22 | 21 | 13 | 13 | 3 | 3 | 5083.61 |
| i05.3.2.4.1_10 | 21 | 26 | 27 | 19 | 20 | 1 | 1 | time limit |
| i05.3.2.4.4_05 | 19 | 25 | 25 | 7 | 7 | 1 | 1 | 540.07 |
| i05.3.2.4.4_10 | 23 | 41 | 39 | 9 | 9 | 1 | 1 | 371.14 |
| i05.3.2.4.7_05 | 12 | 17 | 19 | 10 | 10 | 1 | 1 | 158.47 |
| i05.3.2.4.7_10 | 25 | 35 | 36 | 11 | 11 | 1 | 1 | 195.08 |
| i05.3.5.2.0_05 | 16 | 22 | 24 | 76 | 50 | 1 | 1 | time limit |
| i05.3.5.2.0_10 | 60 | 60 | 60 | 24 | 21 | 1 | 1 | time limit |
| i05.3.5.2.3_05 | 16 | 19 | 20 | 33 | 29 | 1 | 1 | time limit |
| i05.3.5.2.3_10 | 32 | 41 | 42 | 20 | 20 | 1 | 1 | time limit |
| i05.3.5.2.6_05 | 17 | 21 | 22 | 22 | 20 | 1 | 1 | time limit |
| i05.3.5.2.6_10 | 47 | 47 | 47 | 10 | 10 | 1 | 1 | 4237.13 |
| i05.3.5.2.9_05 | 14 | 23 | 23 | 9 | 9 | 1 | 1 | time limit |
| i05.3.5.2.9_10 | 31 | 40 | 41 | 12 | 12 | 1 | 1 | time limit |
| i05.3.5.3.2_05 | 15 | 20 | 22 | 10 | 10 | 1 | 1 | time limit |
| i05.3.5.3.2_10 | 96 | 96 | 96 | 24 | 22 | 1 | 1 | time limit |
| i05.3.5.3.5_05 | 16 | 20 | 21 | 13 | 13 | 1 | 1 | time limit |
| i05.3.5.3.5_10 | 60 | 61 | 60 | 9 | 9 | 3 | 3 | 5548.7 |
| i05.3.5.3.8_05 | 19 | 20 | 21 | 15 | 16 | 1 | 1 | time limit |
| i05.3.5.3.8_10 | 52 | 52 | 52 | 8 | 8 | 1 | 1 | time limit |
| i05.3.5.4.1_05 | 16 | 27 | 24 | 8 | 8 | 1 | 3 | 3572.45 |
| i05.3.5.4.1_10 | 23 | 31 | 35 | 55 | 40 | 1 | 3 | time limit |
| i05.3.5.4.4_05 | 19 | 22 | 23 | 22 | 20 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i05.3.5.4.4_10 | 23 | 31 | 32 | 15 | 16 | 1 | 1 | time limit |
| i05.3.5.4.7_05 | 15 | 22 | 19 | 19 | 19 | 1 | 1 | time limit |
| i05.3.5.4.7_10 | 34 | 47 | 50 | 9 | 9 | 1 | 1 | 4915.6 |
| i05.3.8.2.0_05 | 27 | 27 | 27 | 9 | 9 | 1 | 1 | time limit |
| i05.3.8.2.0_10 | 55 | 55 | 55 | 9 | 9 | 1 | 1 | time limit |
| i05.3.8.2.3_05 | 22 | 24 | 25 | 32 | 26 | 2 | 1 | time limit |
| i05.3.8.2.3_10 | 38 | 38 | 38 | 12 | 12 | 1 | 1 | time limit |
| i05.3.8.2.6_05 | 0 | 24 | 24 | 221 | 166 | 0 | 1 | time limit |
| i05.3.8.2.6_10 | 52 | 52 | 52 | 10 | 10 | 1 | 1 | time limit |
| i05.3.8.2.9_05 | 42 | 42 | 42 | 13 | 13 | 1 | 1 | time limit |
| i05.3.8.2.9_10 | 33 | 42 | 44 | 14 | 14 | 1 | 1 | time limit |
| i05.3.8.3.2_05 | 14 | 18 | 19 | 91 | 59 | 1 | 1 | time limit |
| i05.3.8.3.2_10 | 41 | 41 | 41 | 11 | 11 | 1 | 1 | time limit |
| i05.3.8.3.5_05 | 36 | 36 | 36 | 8 | 8 | 1 | 1 | time limit |
| i05.3.8.3.5_10 | 40 | 40 | 45 | 9 | 9 | 1 | 1 | time limit |
| i05.3.8.3.8_05 | 25 | 26 | 31 | 8 | 8 | 3 | 3 | time limit |
| i05.3.8.3.8_10 | 26 | 36 | 37 | 12 | 12 | 1 | 1 | time limit |
| i05.3.8.4.1_05 | 14 | 19 | 19 | 49 | 33 | 1 | 1 | time limit |
| i05.3.8.4.1_10 | 26 | 34 | 35 | 19 | 19 | 1 | 1 | time limit |
| i05.3.8.4.4_05 | 11 | 16 | 16 | 44 | 30 | 1 | 1 | time limit |
| i05.3.8.4.4_10 | 23 | 38 | 38 | 12 | 12 | 1 | 1 | time limit |
| i05.3.8.4.7_05 | 16 | 22 | 23 | 9 | 9 | 1 | 1 | time limit |
| i05.3.8.4.7_10 | 24 | 44 | 42 | 20 | 20 | 1 | 1 | time limit |
| i05.4.1.2.0_05 | 23 | 23 | 23 | 14 | 14 | 1 | 1 | time limit |
| i05.4.1.2.0_10 | 39 | 41 | 43 | 10 | 10 | 1 | 1 | time limit |
| i05.4.1.2.3_05 | 24 | 24 | 24 | 10 | 10 | 1 | 1 | time limit |
| i05.4.1.2.3_10 | 52 | 52 | 54 | 10 | 10 | 1 | 1 | time limit |
| i05.4.1.2.6_05 | 14 | 18 | 20 | 23 | 23 | 1 | 1 | time limit |
| i05.4.1.2.6_10 | 45 | 45 | 45 | 44 | 34 | 1 | 1 | time limit |
| i05.4.1.2.9_05 | 16 | 21 | 21 | 38 | 32 | 1 | 1 | time limit |
| i05.4.1.2.9_10 | 51 | 51 | 51 | 16 | 16 | 2 | 2 | time limit |
| i05.4.1.3.2_05 | 18 | 25 | 27 | 15 | 15 | 1 | 1 | time limit |
| i05.4.1.3.2_10 | 61 | 61 | 61 | 9 | 9 | 1 | 1 | time limit |
| i05.4.1.3.5_05 | 12 | 16 | 17 | 35 | 26 | 1 | 1 | time limit |
| i05.4.1.3.5_10 | 26 | 35 | 36 | 24 | 21 | 1 | 1 | time limit |
| i05.4.1.3.8_05 | 19 | 19 | 21 | 10 | 10 | 1 | 1 | time limit |
| i05.4.1.3.8_10 | 39 | 39 | 40 | 9 | 9 | 1 | 1 | time limit |
| i05.4.1.4.1_05 | 14 | 19 | 20 | 24 | 24 | 1 | 12 | time limit |
| i05.4.1.4.1_10 | 32 | 42 | 43 | 36 | 28 | 1 | 8 | time limit |
| i05.4.1.4.4_05 | 12 | 15 | 15 | 11 | 11 | 1 | 1 | time limit |
| i05.4.1.4.4_10 | 26 | 37 | 39 | 9 | 9 | 1 | 1 | time limit |
| i05.4.1.4.7_05 | 11 | 16 | 17 | 9 | 9 | 1 | 1 | time limit |
| i05.4.1.4.7_10 | 33 | 45 | 45 | 23 | 22 | 1 | 1 | time limit |
| i05.4.4.2.0_05 | 23 | 23 | 23 | 10 | 10 | 1 | 1 | time limit |
| i05.4.4.2.0_10 | 37 | 37 | 37 | 31 | 24 | 1 | 1 | time limit |
| i05.4.4.2.3_05 | 19 | 19 | 19 | 11 | 11 | 1 | 1 | time limit |
| i05.4.4.2.3_10 | 30 | 43 | 43 | 61 | 40 | 1 | 1 | time limit |
| i05.4.4.2.6_05 | 23 | 23 | 23 | 27 | 23 | 1 | 2 | time limit |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------------------|------------------------------|-------------------|-------------------|------------|
| i05.4.4.2.6_10 | 70 | 70 | 70 | 9 | 9 | 1 | 1 | time limit |
| i05.4.4.2.9_05 | 27 | 27 | 27 | 14 | 14 | 1 | 1 | time limit |
| i05.4.4.2.9_10 | 30 | 41 | 40 | 63 | 40 | 1 | 4 | time limit |
| i05.4.4.3.2_05 | 25 | 25 | 25 | 16 | 16 | 1 | 1 | time limit |
| i05.4.4.3.2_10 | 64 | 64 | 64 | 10 | 10 | 1 | 1 | time limit |
| i05.4.4.3.5_05 | 21 | 21 | 21 | 10 | 10 | 1 | 1 | time limit |
| i05.4.4.3.5_10 | 43 | 44 | 43 | 10 | 10 | 3 | 3 | time limit |
| i05.4.4.3.8_05 | 21 | 21 | 23 | 81 | 52 | 1 | 1 | time limit |
| i05.4.4.3.8_10 | 29 | 41 | 43 | 19 | 18 | 1 | 1 | time limit |
| i05.4.4.4.1_05 | 13 | 18 | 18 | 13 | 13 | 1 | 1 | time limit |
| i05.4.4.4.1_10 | 37 | 48 | 49 | 9 | 9 | 1 | 1 | time limit |
| i05.4.4.4.4_05 | 14 | 19 | 20 | 12 | 12 | 1 | 1 | time limit |
| i05.4.4.4.4_10 | 27 | 35 | 37 | 14 | 14 | 1 | 4 | time limit |
| i05.4.4.4.7_05 | 13 | 24 | 21 | 10 | 10 | 1 | 3 | time limit |
| i05.4.4.4.7_10 | 30 | 37 | 39 | 10 | 10 | 1 | 1 | time limit |
| i05.4.7.2.0_05 | 11 | 15 | 15 | 18 | 18 | 1 | 1 | time limit |
| i05.4.7.2.0_10 | 30 | 39 | 39 | 12 | 12 | 1 | 1 | time limit |
| i05.4.7.2.3_05 | 33 | 33 | 33 | 8 | 8 | 1 | 1 | time limit |
| i05.4.7.2.3_10 | 33 | 35 | 38 | 119 | 74 | 3 | 2 | time limit |
| i05.4.7.2.6_05 | 11 | 16 | 17 | 92 | 56 | 1 | 1 | time limit |
| i05.4.7.2.6_10 | 0 | 38 | 41 | 148 | 142 | 0 | 1 | time limit |
| i05.4.7.2.9_05 | 18 | 18 | 18 | 16 | 16 | 1 | 1 | time limit |
| i05.4.7.2.9_10 | 67 | 67 | 67 | 30 | 28 | 2 | 1 | time limit |
| i05.4.7.3.2_05 | 33 | 33 | 33 | 54 | 37 | 1 | 1 | time limit |
| i05.4.7.3.2_10 | 26 | 39 | 40 | 13 | 13 | 1 | 1 | time limit |
| i05.4.7.3.5_05 | 16 | 22 | 23 | 94 | 52 | 1 | 1 | time limit |
| i05.4.7.3.5_10 | 31 | 42 | 44 | 12 | 12 | 1 | 1 | time limit |
| i05.4.7.3.8_05 | 13 | 17 | 18 | 18 | 18 | 1 | 1 | time limit |
| i05.4.7.3.8_10 | 31 | 39 | 43 | 13 | 13 | 1 | 1 | time limit |
| i05.4.7.4.1_05 | 13 | 19 | 20 | 12 | 12 | 1 | 6 | time limit |
| i05.4.7.4.1_10 | 25 | 34 | 35 | 24 | 21 | 1 | 1 | time limit |
| i05.4.7.4.4_05 | 12 | 16 | 16 | 13 | 13 | 1 | 3 | time limit |
| i05.4.7.4.4_10 | 27 | 37 | 38 | 10 | 10 | 1 | 3 | time limit |
| i05.4.7.4.7_05 | 15 | 20 | 20 | 11 | 11 | 1 | 1 | time limit |
| i05.4.7.4.7_10 | 35 | 46 | 44 | 75 | 51 | 1 | 10 | time limit |
| i10.2.0.2.0_05 | 38 | 38 | 38 | 37 | 37 | 1 | 1 | 1.34 |
| i10.2.0.2.0_10 | 73 | 73 | 73 | 40 | 40 | 1 | 1 | 2.85 |
| i10.2.0.2.3_05 | 65 | 65 | 65 | 36 | 36 | 1 | 1 | 1.46 |
| i10.2.0.2.3_10 | 94 | 94 | 95 | 40 | 40 | 1 | 1 | 2.71 |
| i10.2.0.2.6_05 | 32 | 47 | 50 | 39 | 39 | 1 | 1 | 5.78 |
| i10.2.0.2.6_10 | 87 | 87 | 87 | 38 | 38 | 1 | 1 | 2.41 |
| i10.2.0.2.9_05 | 57 | 57 | 57 | 36 | 36 | 1 | 1 | 2.28 |
| i10.2.0.2.9_10 | 127 | 127 | 127 | 38 | 38 | 1 | 1 | 5.15 |
| i10.2.0.3.2_05 | 82 | 82 | 82 | 30 | 30 | 1 | 1 | 1.68 |
| i10.2.0.3.2_10 | 101 | 101 | 101 | 35 | 35 | 1 | 1 | 4.27 |
| i10.2.0.3.5_05 | 29 | 29 | 29 | 43 | 43 | 1 | 1 | 2.67 |
| i10.2.0.3.5_10 | 60 | 87 | 91 | 38 | 38 | 1 | 1 | 7.22 |
| i10.2.0.3.8_05 | 24 | 34 | 36 | 41 | 41 | 1 | 1 | 4.54 |

| instance | primal | dual | GRASP | ncp _p | ncp _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------|------------------|-------------------|-------------------|------------|
| i10.2.0.3.8_10 | 41 | 57 | 58 | 40 | 40 | 1 | 1 | 2.57 |
| i10.2.0.4.1_05 | 22 | 30 | 31 | 40 | 40 | 1 | 13 | 4.66 |
| i10.2.0.4.1_10 | 46 | 71 | 74 | 38 | 38 | 1 | 1 | 1.31 |
| i10.2.0.4.4_05 | 21 | 28 | 31 | 40 | 40 | 1 | 1 | 2.55 |
| i10.2.0.4.4_10 | 34 | 60 | 58 | 38 | 38 | 1 | 1 | 1.46 |
| i10.2.0.4.7_05 | 22 | 30 | 34 | 39 | 39 | 1 | 1 | 2.24 |
| i10.2.0.4.7_10 | 60 | 79 | 89 | 33 | 33 | 1 | 27 | 9.69 |
| i10.2.3.2.0_05 | 48 | 48 | 48 | 42 | 42 | 1 | 1 | 15.38 |
| i10.2.3.2.0_10 | 63 | 63 | 63 | 37 | 37 | 1 | 1 | 2.78 |
| i10.2.3.2.3_05 | 57 | 57 | 57 | 37 | 37 | 1 | 1 | 3.81 |
| i10.2.3.2.3_10 | 78 | 104 | 101 | 46 | 46 | 1 | 1 | 18.56 |
| i10.2.3.2.6_05 | 53 | 53 | 53 | 40 | 40 | 1 | 1 | 5.15 |
| i10.2.3.2.6_10 | 100 | 100 | 100 | 37 | 37 | 1 | 1 | 4.19 |
| i10.2.3.2.9_05 | 44 | 44 | 44 | 45 | 45 | 1 | 1 | 12.16 |
| i10.2.3.2.9_10 | 132 | 132 | 132 | 33 | 33 | 1 | 1 | 5.43 |
| i10.2.3.3.2_05 | 54 | 54 | 54 | 45 | 45 | 1 | 1 | 12.32 |
| i10.2.3.3.2_10 | 43 | 62 | 66 | 48 | 48 | 1 | 1 | 21.19 |
| i10.2.3.3.5_05 | 20 | 27 | 30 | 41 | 41 | 1 | 1 | 4.35 |
| i10.2.3.3.5_10 | 58 | 84 | 85 | 44 | 44 | 1 | 1 | 13.22 |
| i10.2.3.3.8_05 | 40 | 40 | 42 | 33 | 33 | 1 | 1 | 3.92 |
| i10.2.3.3.8_10 | 53 | 73 | 92 | 66 | 66 | 1 | 126 | 44.93 |
| i10.2.3.4.1_05 | 21 | 28 | 31 | 43 | 43 | 1 | 4 | 6.62 |
| i10.2.3.4.1_10 | 49 | 71 | 76 | 37 | 37 | 1 | 4 | 3.15 |
| i10.2.3.4.4_05 | 25 | 33 | 35 | 39 | 39 | 1 | 7 | 10.27 |
| i10.2.3.4.4_10 | 42 | 56 | 60 | 40 | 40 | 1 | 1 | 4.58 |
| i10.2.3.4.7_05 | 21 | 39 | 37 | 36 | 36 | 1 | 9 | 2.97 |
| i10.2.3.4.7_10 | 42 | 57 | 64 | 48 | 48 | 1 | 1 | 20.5 |
| i10.2.6.2.0_05 | 27 | 36 | 38 | 65 | 66 | 1 | 1 | time limit |
| i10.2.6.2.0_10 | 73 | 73 | 73 | 43 | 43 | 1 | 1 | 12.76 |
| i10.2.6.2.3_05 | 45 | 45 | 45 | 29 | 29 | 1 | 1 | 7.06 |
| i10.2.6.2.3_10 | 93 | 93 | 117 | 32 | 32 | 1 | 1 | 14.65 |
| i10.2.6.2.6_05 | 25 | 38 | 40 | 93 | 91 | 1 | 1 | time limit |
| i10.2.6.2.6_10 | 120 | 120 | 120 | 36 | 36 | 1 | 1 | 9.17 |
| i10.2.6.2.9_05 | 25 | 34 | 36 | 47 | 48 | 1 | 1 | 703.77 |
| i10.2.6.2.9_10 | 99 | 99 | 99 | 42 | 43 | 1 | 1 | 65.92 |
| i10.2.6.3.2_05 | 36 | 39 | 42 | 36 | 36 | 1 | 15 | 8.34 |
| i10.2.6.3.2_10 | 52 | 74 | 80 | 36 | 36 | 1 | 15 | 12.23 |
| i10.2.6.3.5_05 | 46 | 53 | 57 | 34 | 34 | 1 | 1 | 23.12 |
| i10.2.6.3.5_10 | 92 | 92 | 92 | 43 | 43 | 1 | 1 | 4337.22 |
| i10.2.6.3.8_05 | 33 | 33 | 39 | 45 | 45 | 7 | 7 | 19.32 |
| i10.2.6.3.8_10 | 44 | 57 | 59 | 40 | 40 | 1 | 7 | 9.72 |
| i10.2.6.4.1_05 | 29 | 40 | 45 | 131 | 143 | 1 | 1 | time limit |
| i10.2.6.4.1_10 | 57 | 80 | 84 | 35 | 35 | 1 | 1 | 13.81 |
| i10.2.6.4.4_05 | 21 | 39 | 39 | 121 | 134 | 1 | 1 | time limit |
| i10.2.6.4.4_10 | 42 | 59 | 64 | 42 | 42 | 1 | 22 | 18.1 |
| i10.2.6.4.7_05 | 25 | 37 | 38 | 36 | 36 | 1 | 4 | 7.81 |
| i10.2.6.4.7_10 | 41 | 55 | 61 | 53 | 54 | 1 | 85 | 68.51 |
| i10.2.9.2.0_05 | 22 | 33 | 36 | 117 | 154 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------------------|------------------------------|-------------------|-------------------|------------|
| i10.2.9.2.0_10 | 71 | 71 | 71 | 40 | 40 | 1 | 1 | 34.57 |
| i10.2.9.2.3_05 | 55 | 55 | 55 | 39 | 39 | 1 | 1 | 20.47 |
| i10.2.9.2.3_10 | 68 | 243 | 83 | 42 | 42 | 1 | 1 | 34.0 |
| i10.2.9.2.6_05 | 54 | 54 | 54 | 59 | 66 | 1 | 1 | time limit |
| i10.2.9.2.6_10 | 50 | 72 | 79 | 84 | 84 | 1 | 1 | 435.3 |
| i10.2.9.2.9_05 | 24 | 33 | 36 | 66 | 67 | 1 | 1 | 5081.95 |
| i10.2.9.2.9_10 | 89 | 89 | 89 | 52 | 53 | 1 | 1 | 1103.67 |
| i10.2.9.3.2_05 | 22 | 32 | 34 | 42 | 42 | 1 | 1 | 31.28 |
| i10.2.9.3.2_10 | 53 | 63 | 71 | 45 | 46 | 1 | 1 | 88.65 |
| i10.2.9.3.5_05 | 23 | 32 | 33 | 38 | 38 | 1 | 1 | 16.34 |
| i10.2.9.3.5_10 | 38 | 53 | 61 | 46 | 46 | 1 | 1 | 50.29 |
| i10.2.9.3.8_05 | 26 | 35 | 37 | 87 | 103 | 1 | 1 | time limit |
| i10.2.9.3.8_10 | 51 | 67 | 72 | 72 | 74 | 1 | 14 | 526.35 |
| i10.2.9.4.1_05 | 24 | 31 | 32 | 42 | 42 | 1 | 19 | 36.05 |
| i10.2.9.4.1_10 | 41 | 59 | 71 | 52 | 54 | 1 | 4 | 229.01 |
| i10.2.9.4.4_05 | 21 | 29 | 30 | 63 | 66 | 1 | 1 | 3472.1 |
| i10.2.9.4.4_10 | 45 | 69 | 74 | 37 | 37 | 1 | 1 | 42.18 |
| i10.2.9.4.7_05 | 26 | 33 | 36 | 38 | 38 | 1 | 1 | 24.8 |
| i10.2.9.4.7_10 | 43 | 60 | 64 | 40 | 40 | 1 | 1 | 58.26 |
| i10.3.2.2.0_05 | 36 | 36 | 36 | 44 | 44 | 1 | 1 | 667.34 |
| i10.3.2.2.0_10 | 64 | 65 | 69 | 60 | 60 | 1 | 1 | time limit |
| i10.3.2.2.3_05 | 22 | 36 | 37 | 84 | 93 | 1 | 1 | time limit |
| i10.3.2.2.3_10 | 126 | 126 | 126 | 38 | 38 | 1 | 1 | 194.41 |
| i10.3.2.2.6_05 | 41 | 41 | 41 | 161 | 130 | 1 | 1 | time limit |
| i10.3.2.2.6_10 | 52 | 59 | 70 | 65 | 66 | 1 | 1 | time limit |
| i10.3.2.2.9_05 | 20 | 30 | 32 | 60 | 60 | 1 | 1 | 708.92 |
| i10.3.2.2.9_10 | 66 | 66 | 66 | 51 | 51 | 1 | 1 | 1282.59 |
| i10.3.2.3.2_05 | 25 | 35 | 38 | 57 | 59 | 1 | 1 | 2195.88 |
| i10.3.2.3.2_10 | 38 | 61 | 67 | 50 | 52 | 1 | 3 | time limit |
| i10.3.2.3.5_05 | 23 | 31 | 34 | 58 | 60 | 1 | 31 | 2881.99 |
| i10.3.2.3.5_10 | 42 | 56 | 64 | 71 | 73 | 1 | 1 | time limit |
| i10.3.2.3.8_05 | 20 | 27 | 29 | 53 | 54 | 1 | 1 | time limit |
| i10.3.2.3.8_10 | 43 | 58 | 61 | 76 | 80 | 1 | 1 | time limit |
| i10.3.2.4.1_05 | 22 | 30 | 30 | 71 | 71 | 1 | 12 | 4442.31 |
| i10.3.2.4.1_10 | 31 | 62 | 55 | 42 | 42 | 1 | 1 | 358.86 |
| i10.3.2.4.4_05 | 20 | 27 | 29 | 69 | 72 | 1 | 201 | time limit |
| i10.3.2.4.7_05 | 20 | 27 | 30 | 58 | 58 | 1 | 1 | time limit |
| i10.3.2.4.7_10 | 49 | 66 | 67 | 283 | 162 | 1 | 16 | time limit |
| i10.3.5.2.0_05 | 0 | 38 | 41 | 304 | 402 | 0 | 8 | time limit |
| i10.3.5.2.0_10 | 96 | 96 | 96 | 60 | 60 | 1 | 1 | time limit |
| i10.3.5.2.3_05 | 46 | 46 | 46 | 362 | 225 | 1 | 25 | time limit |
| i10.3.5.2.3_10 | 123 | 123 | 123 | 34 | 34 | 1 | 1 | time limit |
| i10.3.5.2.6_05 | 36 | 36 | 39 | 44 | 44 | 1 | 1 | time limit |
| i10.3.5.2.6_10 | 56 | 56 | 56 | 235 | 160 | 1 | 1 | time limit |
| i10.3.5.2.9_05 | 55 | 55 | 55 | 41 | 41 | 1 | 1 | time limit |
| i10.3.5.2.9_10 | 89 | 89 | 89 | 136 | 100 | 1 | 1 | time limit |
| i10.3.5.3.2_05 | 51 | 51 | 51 | 34 | 34 | 1 | 1 | time limit |
| i10.3.5.3.2_10 | 45 | 61 | 66 | 51 | 51 | 1 | 1 | time limit |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|------|-------|------------------------------|------------------------------|-------------------|-------------------|------------|
| i10.3.5.3.5_05 | 42 | 43 | 42 | 136 | 133 | 3 | 3 | time limit |
| i10.3.5.3.5_10 | 52 | 54 | 58 | 303 | 182 | 1 | 1 | time limit |
| i10.3.5.3.8_05 | 32 | 32 | 32 | 59 | 59 | 1 | 1 | time limit |
| i10.3.5.3.8_10 | 48 | 70 | 73 | 35 | 35 | 1 | 7 | time limit |
| i10.3.5.4.1_05 | 23 | 32 | 33 | 83 | 75 | 1 | 26 | time limit |
| i10.3.5.4.1_10 | 43 | 58 | 60 | 57 | 57 | 1 | 1 | time limit |
| i10.3.5.4.4_05 | 25 | 35 | 36 | 41 | 41 | 1 | 7 | time limit |
| i10.3.5.4.7_05 | 23 | 31 | 32 | 232 | 167 | 1 | 1 | time limit |
| i10.3.5.4.7_10 | 42 | 57 | 62 | 75 | 75 | 1 | 1 | time limit |
| i10.3.8.2.0_05 | 0 | 34 | 36 | 213 | 721 | 0 | 1 | time limit |
| i10.3.8.2.0_10 | 87 | 87 | 89 | 216 | 149 | 1 | 1 | time limit |
| i10.3.8.2.3_05 | 25 | 43 | 41 | 68 | 68 | 1 | 1 | time limit |
| i10.3.8.2.3_10 | 78 | 78 | 78 | 51 | 51 | 1 | 1 | time limit |
| i10.3.8.2.6_05 | 26 | 32 | 33 | 88 | 85 | 1 | 1 | time limit |
| i10.3.8.2.6_10 | 64 | 81 | 85 | 99 | 77 | 1 | 1 | time limit |
| i10.3.8.2.9_10 | 0 | 96 | 96 | 228 | 314 | 0 | 1 | time limit |
| i10.3.8.3.2_05 | 31 | 31 | 33 | 40 | 40 | 1 | 1 | time limit |
| i10.3.8.3.2_10 | 0 | 78 | 82 | 229 | 179 | 0 | 1 | time limit |
| i10.3.8.3.5_05 | 0 | 34 | 34 | 265 | 202 | 0 | 1 | time limit |
| i10.3.8.3.5_10 | 55 | 91 | 98 | 97 | 82 | 1 | 3 | time limit |
| i10.3.8.3.8_05 | 0 | 32 | 39 | 243 | 256 | 0 | 1 | time limit |
| i10.3.8.3.8_10 | 0 | 59 | 65 | 238 | 233 | 0 | 13 | time limit |
| i10.3.8.4.1_05 | 0 | 27 | 29 | 240 | 180 | 0 | 9 | time limit |
| i10.3.8.4.1_10 | 46 | 68 | 67 | 54 | 54 | 1 | 1 | time limit |
| i10.3.8.4.4_05 | 21 | 36 | 34 | 188 | 137 | 1 | 1 | time limit |
| i10.3.8.4.4_10 | 0 | 50 | 53 | 253 | 266 | 0 | 62 | time limit |
| i10.3.8.4.7_05 | 21 | 41 | 36 | 102 | 83 | 1 | 5 | time limit |
| i10.3.8.4.7_10 | 46 | 61 | 68 | 75 | 75 | 1 | 5 | time limit |
| i10.4.1.2.0_05 | 41 | 41 | 41 | 63 | 63 | 1 | 1 | time limit |
| i10.4.1.2.0_10 | 0 | 62 | 67 | 218 | 312 | 0 | 2 | time limit |
| i10.4.1.2.3_05 | 56 | 56 | 45 | 58 | 58 | 1 | 1 | time limit |
| i10.4.1.2.3_10 | 91 | 91 | 91 | 52 | 52 | 1 | 1 | time limit |
| i10.4.1.2.6_05 | 31 | 54 | 54 | 57 | 57 | 1 | 1 | time limit |
| i10.4.1.2.6_10 | 60 | 60 | 60 | 144 | 111 | 1 | 6 | time limit |
| i10.4.1.2.9_05 | 33 | 33 | 33 | 45 | 45 | 1 | 1 | time limit |
| i10.4.1.2.9_10 | 91 | 91 | 91 | 67 | 66 | 1 | 1 | time limit |
| i10.4.1.3.2_05 | 17 | 23 | 26 | 148 | 112 | 1 | 6 | time limit |
| i10.4.1.3.2_10 | 0 | 70 | 77 | 211 | 357 | 0 | 23 | time limit |
| i10.4.1.3.5_05 | 25 | 37 | 39 | 70 | 70 | 1 | 1 | time limit |
| i10.4.1.3.5_10 | 43 | 60 | 62 | 54 | 54 | 1 | 1 | time limit |
| i10.4.1.3.8_05 | 26 | 37 | 39 | 151 | 110 | 1 | 1 | time limit |
| i10.4.1.3.8_10 | 50 | 69 | 71 | 117 | 96 | 1 | 1 | time limit |
| i10.4.1.4.1_05 | 22 | 28 | 32 | 72 | 72 | 1 | 84 | time limit |
| i10.4.1.4.1_10 | 54 | 90 | 92 | 142 | 111 | 1 | 22 | time limit |
| i10.4.1.4.4_05 | 0 | 30 | 32 | 213 | 234 | 0 | 89 | time limit |
| i10.4.1.4.4_10 | 40 | 54 | 57 | 123 | 94 | 1 | 1 | time limit |
| i10.4.1.4.7_05 | 24 | 31 | 32 | 70 | 68 | 1 | 63 | time limit |
| i10.4.1.4.7_10 | 54 | 78 | 85 | 45 | 45 | 1 | 31 | time limit |

| instance | primal | dual | GRASP | n _{cp} _p | n _{cp} _d | tree _p | tree _d | enum time |
|----------------|--------|-------|-------|------------------------------|------------------------------|-------------------|-------------------|------------|
| i10.4.4.2.0_05 | 0 | 31 | 33 | 194 | 271 | 0 | 1 | time limit |
| i10.4.4.2.0_10 | 0 | 83 | 87 | 182 | 264 | 0 | 1 | time limit |
| i10.4.4.2.3_05 | 0 | 33 | 34 | 187 | 182 | 0 | 1 | time limit |
| i10.4.4.2.3_10 | 68 | 68 | 68 | 113 | 90 | 1 | 1 | time limit |
| i10.4.4.2.6_05 | 25 | 39 | 40 | 70 | 68 | 1 | 1 | time limit |
| i10.4.4.2.6_10 | 70 | 70 | 70 | 191 | 138 | 1 | 1 | time limit |
| i10.4.4.2.9_05 | 45 | 45 | 51 | 43 | 43 | 1 | 2 | time limit |
| i10.4.4.2.9_10 | 77 | 77 | 77 | 191 | 138 | 1 | 1 | time limit |
| i10.4.4.3.2_05 | 35 | 35 | 35 | 52 | 52 | 1 | 1 | time limit |
| i10.4.4.3.2_10 | 69 | 69 | 69 | 187 | 138 | 1 | 1 | time limit |
| i10.4.4.3.5_05 | 0 | 1e+20 | 33 | 173 | 2076 | 0 | 0 | time limit |
| i10.4.4.3.5_10 | 0 | 77 | 76 | 183 | 197 | 0 | 9 | time limit |
| i10.4.4.3.8_05 | 24 | 32 | 33 | 178 | 128 | 1 | 1 | time limit |
| i10.4.4.3.8_10 | 49 | 70 | 77 | 171 | 133 | 1 | 1 | time limit |
| i10.4.4.4.1_05 | 19 | 29 | 31 | 92 | 80 | 1 | 1 | time limit |
| i10.4.4.4.1_10 | 43 | 60 | 61 | 39 | 39 | 1 | 2 | time limit |
| i10.4.4.4.4_05 | 0 | 33 | 41 | 173 | 411 | 0 | 1 | time limit |
| i10.4.4.4.7_05 | 16 | 28 | 30 | 112 | 94 | 1 | 12 | time limit |
| i10.4.4.4.7_10 | 43 | 60 | 63 | 47 | 47 | 1 | 1 | time limit |
| i10.4.7.2.0_05 | 36 | 41 | 41 | 52 | 52 | 1 | 1 | time limit |
| i10.4.7.2.0_10 | 103 | 103 | 103 | 159 | 111 | 1 | 1 | time limit |
| i10.4.7.2.3_05 | 47 | 47 | 48 | 130 | 99 | 1 | 5 | time limit |
| i10.4.7.2.3_10 | 0 | 76 | 76 | 162 | 137 | 0 | 1 | time limit |
| i10.4.7.2.6_05 | 32 | 39 | 39 | 78 | 78 | 1 | 1 | time limit |
| i10.4.7.2.6_10 | 0 | 63 | 64 | 159 | 201 | 0 | 1 | time limit |
| i10.4.7.2.9_05 | 0 | 37 | 39 | 158 | 167 | 0 | 1 | time limit |
| i10.4.7.2.9_10 | 40 | 65 | 72 | 48 | 48 | 1 | 3 | time limit |
| i10.4.7.3.2_05 | 0 | 34 | 39 | 143 | 585 | 0 | 1 | time limit |
| i10.4.7.3.2_10 | 52 | 75 | 78 | 65 | 64 | 1 | 1 | time limit |
| i10.4.7.3.5_05 | 0 | 30 | 31 | 158 | 398 | 0 | 1 | time limit |
| i10.4.7.3.5_10 | 0 | 50 | 54 | 161 | 187 | 0 | 1 | time limit |
| i10.4.7.3.8_05 | 21 | 29 | 32 | 44 | 44 | 1 | 1 | time limit |
| i10.4.7.3.8_10 | 37 | 56 | 62 | 114 | 90 | 1 | 1 | time limit |
| i10.4.7.4.1_10 | 0 | 63 | 68 | 146 | 190 | 0 | 16 | time limit |
| i10.4.7.4.4_05 | 28 | 38 | 40 | 44 | 44 | 1 | 1 | time limit |
| i10.4.7.4.4_10 | 41 | 56 | 59 | 81 | 81 | 1 | 1 | time limit |
| i10.4.7.4.7_05 | 0 | 32 | 34 | 187 | 151 | 0 | 10 | time limit |
| i10.4.7.4.7_10 | 0 | 68 | 78 | 164 | 204 | 0 | 76 | time limit |